Intents and Intent Filters

Akhilesh Tyagi
Communicating Activities

- Many apps have **multiple activities**.
  - Example: In an address book app, the main activity is a list of contacts, and clicking on a contact goes to another activity for viewing details.
  - An activity A can launch another activity B in response to an event.
  - The activity A can pass data to B.
  - The second activity B can send data back to A when it is done.
Activities in Manifest

- Every activity has an entry in project's AndroidManifest.xml, added automatically by Android Studio:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myusername.myapplication">
    <application android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            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=".SecondActivity"
            android:parentActivityName=".SecondActivity"
            android:label="@string/title_activity_second">
            <meta-data android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.myusername.myapplication.MainActivity"/>
        </activity>
    </application>
</manifest>
```

Marty Stepps, CS 193A, Stanford
Intents

• **intent**: a bridge between activities; a way for one activity to invoke another
  – the activity can be in the same app or in a different app
  – can store extra data to pass as "parameters" to that activity
  – second activity can "**return**" information back to the caller if needed
Intents

• Intents form dynamic communication channels between activities, within activities, between activities and services & broadcast events.

• Allows late and dynamic binding between program components.

• Message passing mechanism
Intent

• An intent is a passive data structure holding an abstract description of an operation to be performed.
• Contains an `<action, category, data>`
• They are most commonly used to launch different activities within an application.
Intent Object

• An intent object is a bundle of information.
• It contains information of interest to the component that receives the intent (such as the action to be taken and the data to act on).
• It can also contain information of interest to the Android system.
Intent use scenarios

• One of the most common uses for Intents is to start new Activities, either explicitly (by specifying the class) or implicitly (by requesting an action be performed on a piece of data).

• Intents can also be used to broadcast messages across the system. Registered broadcast receivers are matched to provide “broadcast handling”.

• Using Intents to propagate actions — even within the same application — is a fundamental Android design principle. It encourages the decoupling of components, to allow the seamless replacement of application elements.
Intent Object

• The main information that can be contained in an intent object:
  – **Component name**: The name of the component that should handle the intent.
  – **Action**: An action to be performed such as initiating a phone call in an activity or handling a low battery warning in a broadcast.
  – **Data**: Data to be acted on.
  – **Category**: Additional information about the component handling the intent such as a widget, home, preference.
  – **Extras**: Key value pairs containing additional information that should be passed to the component.
  – **Flags**: These are contained in the intent API and have many different functions for the system on how to handle the intent.
Using Intents to Launch Activities

• To open a different application screen (Activity) in your application, call startActivity, passing in an Intent

```java
startActivity(myIntent);
```

• The Intent can either explicitly specify the class to open, or include an action that the target should perform. Run time will choose the *implicit* Activity to open with "Intent resolution."

```java
```
Intent Resolution

• Intents can be divided into two groups:
  – Explicit intents: These designate a component by name and are generally used locally for internal messages.
  – Implicit intents: Do not name a target and are generally used by outside applications.

• For implicit intents the Android system will search components using intent filters to find the best match.
Starting an Activity (*fire & forget*)

```java
Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);
startActivity(intent);
```

- After calling `startActivity`, the new Activity (in this example, `MyOtherActivity`) will be created and become visible and active, moving to the top of the Activity stack.

- Calling `finish` programmatically on the new Activity will close it and remove it from the stack. Alternatively, users can navigate to the previous Activity using the device’s Back button.
Implicit Intents and Late Runtime Binding

• *Implicit Intents* are a mechanism that lets anonymous application components service action requests.

• implicit Intent to use with `startActivity`, specifies an action to perform and, optionally, supplies the data on which to perform that action.

```java
if (foundPhoneNumber) {
    Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:555-2368");
    startActivity(intent);
}
```
Extracting extra data

- In the second activity that was invoked, you can grab any extra data that was passed to it by the calling activity.
  - You can access the Intent that spawned you by calling getIntent.
  - The Intent has methods like getExtra, getIntExtra, getStringExtra, etc. to extract any data that was stored inside the intent.

```java
public class SecondActivity extends Activity {
    ...
    public void onCreate(Bundle savedState) {
        super.onCreate(savedState);
        setContentView(R.layout.activity_second);
        Intent intent = getIntent();
        String extra = intent.getExtra("name");
        ...
    }
}
```
Explicit Intent with Data/Arguments

```
Intent i = new Intent(this, ActivityB.class); i.putExtra("myString1", "This is a message for ActivityB");
i.putExtra("myInt1", 100);
startActivity(i);
```

- The data is received at the target activity as part of a Bundle object which can be obtained via a call to `getIntent().getExtras()`.
- ActivityB retrieves the data:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    String myString = extras.getString("myString1");
    int myInt = extras.getInt("myInt1");
}
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ebookfrenzy.Intent1" android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10"/>
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name="ActivityA">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="ActivityB" android:label="ActivityB"/>
    </application>
</manifest>
Another example – button click calling a sub-activity

b4.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent i = new Intent(ctx, WidgetActivity.class);
        Log.i(Global.TAG, "b4 pressed - about to launch sub-activity");

        startActivity(i);
        Log.i(Global.TAG, "b4 pressed - successfully launched sub-activity (startActivity called)");
    }
});
Returning Results from Activities *(return to sender)*

- launch ActivityB as a sub-activity of ActivityA.
- An activity is started as a sub-activity by starting the intent with a call to the `startActivityForResult()` method
- In addition to the intent object, this method is also passed a request code value which can be used to identify the return data when the sub-activity returns.

`startActivityForResult(i, REQUEST_CODE);`
• Subactivity *ActivityB* returns the result with:

```java
public void finish() {
    Intent data = new Intent();
    data.putExtra("returnString1", "Message to parent activity");
    setResult(RESULT_OK, data);
    super.finish();
}
```
ActivityA receiving results

protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    String returnString;
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
        if (data.hasExtra("returnString1")) {
            returnString =
              data.getExtras().getString("returnString1");
        }
    }
}
private static final int SHOW_SUBACTIVITY = 1;
Intent intent = new Intent(this, MyOtherActivity.class);
startActivityForResult(intent, SHOW_SUBACTIVITY);

Implicit to pick a contact:
private static final int PICK_CONTACT_SUBACTIVITY = 2;
Uri uri = Uri.parse("content://contacts/people");
Intent intent = new Intent(Intent.ACTION_PICK, uri);
startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
Returning Results

• When sub-Activity is ready to close, call `setResult` before `finish` to return a result to the calling Activity.

• The `setResult` method takes two parameters: the `result code` and `result payload` represented as an Intent.

• result code is the “result” of running the sub-Activity — generally either `Activity.RESULT_OK` or `Activity.RESULT_CANCELED`
a sub-Activity’s onCreate method shows how an OK button and a Cancel button might return different results to the calling Activity:

```java
Button okButton = (Button) findViewById(R.id.ok_button);
okButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Uri data = Uri.parse("content://books/" + selected_book_id);
        Intent result = new Intent(null, data);
        result.putExtra(IS_INPUT_CORRECT, inputCorrect);
        result.putExtra(SELECTED_POEM, selectedPoem);
    }
});
```
setResult(RESULT_OK, result);
finish();
}
});

Button cancelButton = (Button) findViewById(R.id.cancel_button);
cancelButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        setResult(RESULT_CANCELED, null);
        finish();
    }
});
Handling Subactivity results

• `onActivityResult` handler receives:
  – Request code
  – Result code
  – Data
private static final int SELECT_BOOK = 1;
private static final int SELECT_POEM = 2;

Uri selectedBook = null;
Uri selectedPoem = null;

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {

    super.onActivityResult(requestCode, resultCode, data);

}
switch(requestCode) {
    case (SELECT_BOOK):
        if (resultCode == Activity.RESULT_OK)
            selectedBook = data.getData();
        break;

    case (SELECT_POEM):
        if (resultCode == Activity.RESULT_OK)
            selectedPoem = data.getData();
        break;

    default: break;
}
}
Determining if an Intent will resolve

- Query package manger

```java
if (somethingWeird && itDontLookGood) {
    // Create the implicit Intent to use to start a new Activity.
    Intent intent =
        new Intent(Intent.ACTION_DIAL, Uri.parse("tel:555-2368"));

    // Check if an Activity exists to perform this action.
    PackageManager pm = getPackageManager();
    ComponentName cn = intent.resolveActivity(pm);
    if (cn == null) {
        //log a message or generate a notification
```
Check Google play store if it will resolve your intent

if (cn == null) {

// If there is no Activity available to perform the action
// Check to see if the Market is available.

    Uri marketUri =

    Uri.parse("market://search?q=pname:com.myapp.packageName");

    Intent marketIntent = new Intent(Intent.ACTION_VIEW).setData(marketUri);
If the Market is available, use it to download an application capable of performing the required action. Otherwise log an error.

```java
if (marketIntent.resolveActivity(pm) != null)
    startActivity(marketIntent);
else
    Log.d(TAG, "Market client not available.");
}
else
    startActivity(intent);
}
If no Activity resolves your intent, you can disable the related functionality. Or direct users to an appropriate app in the Play store.
Service Intents

- Service intents are used in a similar way to activities, the only difference being that the second parameter of the new intent is often the service you wish to use.

```java
Intent newService = new Intent(this, MyService.class);
```
Broadcast Intents

• In addition to intents being used to invoke activities, we can also use intents for broadcasts.

• These are generally done in Java Code as they are more dynamic and are often associated with some sort of action.
Broadcast Intents

• You declare a new broadcast in an intent, and can then send that broadcast when necessary.

```java
Intent newBroadcast = new Intent("myBroadcast");

public void SendMyBroadcast()
{
    sendBroadcast(newBroadcast);
}
```
Intent Filters

• To inform the system which implicit intents an Activity can handle. Activities, services, and broadcast receivers can have one or more intent filters.

• Each filter describes a capability of the component, a set of intents that the component is willing to receive.
Intent Filters

• Because intent filters must be seen by the Android system before launching a component; they are often placed in the Android Manifest file instead of Java Code.

• Implicit intents are tested against three filters of a component to see if they are correct.
  – Action
  – Category
  – Data
Intent Filters

• Intent filters are used to declare the entry point of your application.

  <action android:name="android.intent.action.MAIN" />

• As well as link an activity to the launch icon.

  <category android:name="android.intent.category.LAUNCHER" />
<table>
<thead>
<tr>
<th>Action Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_CALL</td>
<td>activity</td>
<td>Initiate a phone call.</td>
</tr>
<tr>
<td>ACTION_EDIT</td>
<td>activity</td>
<td>Display data for the user to edit.</td>
</tr>
<tr>
<td>ACTION_MAIN</td>
<td>activity</td>
<td>Start up as the initial activity of a task, with no data input and no returned output.</td>
</tr>
<tr>
<td>ACTION_SYNC</td>
<td>activity</td>
<td>Synchronize data on a server with data on the mobile device.</td>
</tr>
<tr>
<td>ACTION_BATTERY_LOW</td>
<td>broadcast receiver</td>
<td>A warning that the battery is low.</td>
</tr>
<tr>
<td>ACTION_HEADSET_PLUG</td>
<td>broadcast receiver</td>
<td>A headset has been plugged into the device, or unplugged from it.</td>
</tr>
<tr>
<td>ACTION_SCREEN_ON</td>
<td>broadcast receiver</td>
<td>The screen has been turned on.</td>
</tr>
<tr>
<td>ACTION_TIMEZONE_CHANGED</td>
<td>broadcast receiver</td>
<td>The setting for the time zone has changed.</td>
</tr>
</tbody>
</table>
- ACTION_ANSWER Opens an Activity that handles incoming calls.
- ACTION_CALL Brings up a phone dialer.
- ACTION_DELETE Starts an Activity that lets you delete the entry currently stored at the data URI location.
- ACTION_DIAL Brings up a dialer application with the number to dial prepopulated from the data URI.
- ACTION_EDIT Requests an Activity that can edit the data at the URI.
- ACTION_INSERT Opens an Activity capable of inserting new items into the cursor specified in the data field.
- ACTION_PICK Launches a sub-Activity that lets you pick an item from the URI data.
- **ACTION_SEARCH** Launches the UI for performing a search. Supply the search term as a string.

- **ACTION_SENDTO** Launches an Activity to send a message to the contact.

- **ACTION_SEND** Launches an Activity that sends the specified data.

- **ACTION_VIEW** View asks that the data supplied in the Intent’s URI be viewed in the most reasonable manner.

- **ACTION_WEB_SEARCH** Opens an activity that performs a Web search based on the text supplied in the data URI.
Category

- **category** Use the `android:category` attribute to specify under which circumstances the action should be serviced.

ALTERNATIVE: The alternative category specifies that this action should be available as an alternative to the default action.

BROWSABLE: Specifies an action available from within the browser.
DEFAULT: Set this to make a component the default action. This is also necessary for Activities that are launched using an explicit Intent.

GADGET: By setting the gadget category, you specify that this Activity can run embedded inside another Activity.

HOME: The home Activity is the first Activity displayed when the device starts.

LAUNCHER: Using this category makes an Activity appear in the application launcher.
Data

• Data tag specifies the data types handled by this component.

• `android:host` – specifies a valid host name (iastate.edu)

• `android:mimetype`

• `android:path` – valid path values for URIs

• `android:port` – valid ports for a host
Intent Filter Example

<activity android:name=".EarthquakeDamageViewer"
android:label="View Damage">
<intent-filter>
<action
android:name="com.paad.earthquake.intent.action.SHOW_DAMAGE">
</action>
<category android:name="android.intent.category.DEFAULT"/>
<category
android:name="android.intent.category.ALTERNATIVE_SELECTED"/>
<data android:mimeType="vnd.earthquake.cursor.item/*"/>
</intent-filter>
</activity>
Responding to implicit Intents

- When an application component is started through an implicit Intent, it needs to find the action it is to perform and the data upon which to perform it.

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Intent intent = getIntent();
    String action = intent.getAction();
    Uri data = intent.getData();
}
```
Broadcasting Events with Intents

- Broadcast Intents are used to notify listeners of system or application events, extending the event driven programming model between applications.

- **Action strings:**

  ```java
  public static final String NEW_LIFEFORM_DETECTED =
  "com.paad.action.NEW_LIFEFORM";
  ```
Intent intent = new Intent(NEW_LIFEFORM_DETECTED);
intent.putExtra("lifeformName", lifeformType);
intent.putExtra("longitude", currentLongitude);
intent.putExtra("latitude", currentLatitude);
sendBroadcast(intent);
Listening for Broadcasts with Broadcast Receivers

```java
class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //TODO: React to the Intent received.
    }
}
```

- To enable a Broadcast Receiver, it needs to be registered, either in code or within the application manifest. When registering a Broadcast Receiver, you must use an Intent Filter to specify which Intents it is listening for.
public class LifeformDetectedBroadcastReceiver extends BroadcastReceiver {
public static final String BURN =
"com.paad.alien.action.BURN_IT_WITH_FIRE";
@Override
public void onReceive(Context context, Intent intent) {
// Get the lifeform details from the intent.
    Uri data = intent.getData();
    String type = intent.getStringExtra("type");
    double lat = intent.getDoubleExtra("latitude", 0);
    double lng = intent.getDoubleExtra("longitude", 0);
    Location loc = new Location("gps");
    loc.setLatitude(lat);
    loc.setLongitude(lng);
    if (type.equals("alien")) {
        Intent startIntent = new Intent(BURN, data);
        startIntent.putExtra("latitude", lat);
        StartIntent.putExtra("longitude", lng);
        context.startActivity(startIntent);
    }}}}
Registering Broadcast Receivers in Application Manifest

<receiver
android:name=".LifeformDetectedBroadcastReceiver">
<intent-filter>
   <action
       android:name="com.paad.action.NEW_LIFEFORM"/>
</intent-filter>
</receiver>
Registering Broadcast Receivers in Code

// Create and register the broadcast receiver.
IntentFilter filter = new IntentFilter(NEW_LIFEFORM_DETECTED);
LifeformDetectedBroadcastReceiver r = new LifeformDetectedBroadcastReceiver();
registerReceiver(r, filter);
Broadcast Receivers registered in Code

• If UI is affected by the broadcast intent –
• Register the receiver programmatically.
• The activity must be running for it to respond to the broadcast intent.
private IntentFilter filter =
    new IntentFilter(LifeformDetectedReceiver.NEW_LIFEFORM);

private LifeformDetectedReceiver receiver =
    new LifeformDetectedReceiver();

@Override
public synchronized void onResume() {
    super.onResume();
    // Register the broadcast receiver.
    registerReceiver(receiver, filter);
}

@Override
public synchronized void onPause() {
    // Unregister the receiver
    unregisterReceiver(receiver);
    super.onPause();
}
Ordered Broadcast Intents

• You can enforce an order – sort of build a state-ful response.

    String requiredPermission = “edu.iastate.MY_BROADCAST_PERMISSION”;  
    sendOrderedBroadcast(intent, requiredPermission);

Intent will be delivered to all registered receivers that hold the required permission.
Sticky Intents Broadcast

• When `registerReceiver` occurs, the last intent that was broadcast (is sticky) is sent to the receiver.

```java
IntentFilter battery = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent currentBatteryCharge = registerReceiver(null, battery);
```

• No receiver specified for a sticky intent registration.

• `sendStickyBroadcast(intent);`
Local Broadcast Manager

• Supports event driven communication between your app components
• Reduced broadcast scope is more efficiently supported than global broadcasts
• Based on Android support library
• `LocalBroadcastManager lbm = LocalBroadcastManager.getInstance(this);`
• Register the LB receiver with a broadcast receiver and an intent filter:

```java
lbm.registerReceiver(new BroadcastReceiver(){
    @Override
    public void onReceive(Context context, Intent intent){
        //handle the received broadcast
    }
}, new IntentFilter (LOCAL_ACTION));
```
• Transmit a local broadcast intent:
  `lbm.sendBroadcast(new Intent(LOCAL_ACTION));`