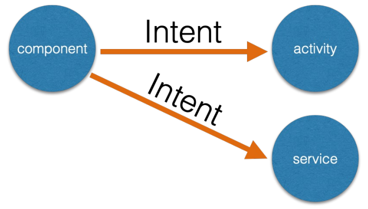
The Intent of Intents

Mobile Computing - Android

Intents

- An intent is a message, sent to an app component (activity, service, broadcast receiver)
- Can be used to:
  - start an **activity**
  - start a **service**
  - **broadcast** information to other apps



Intent Usage

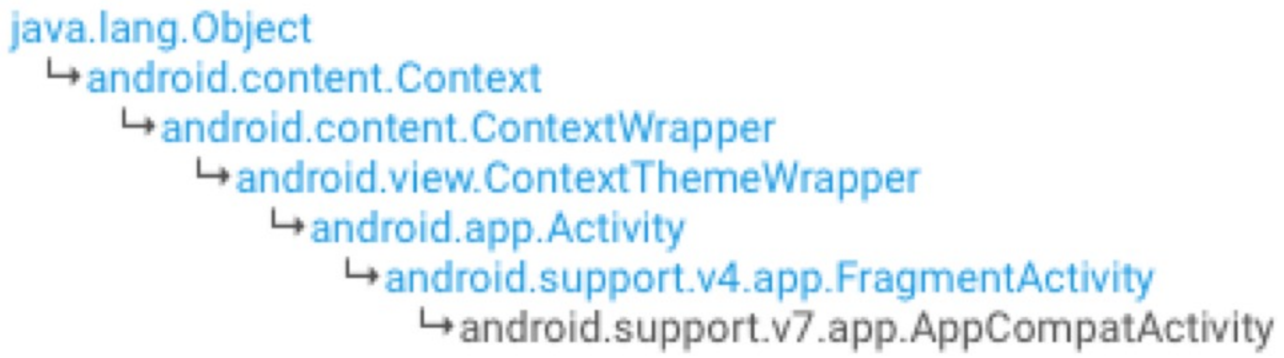
- **Explicit Intents** specify the component to start explicitly, by name. This is usually used to start a component in your own app.
- **Implicit Intents** do not specify a component by name; rather they specify an *action* that needs to be performed. Android then delivers that intent to an app capable of performing that action.

Explicit Intents

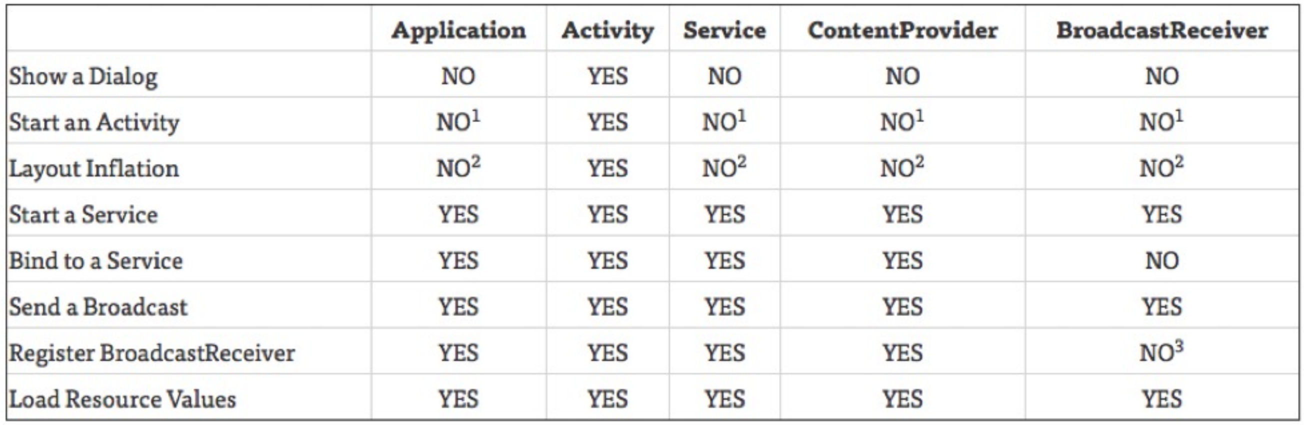
- Name the component (fully qualified class name, including package name)
  - public Intent(Context context, Class classname)
  - Use setClassName(), setComponent() or setClass() if the Intent object has already been instantiated.
- Explicit intents are for a component *in your own app*
  - e.g., switch from one Activity to another (essentially, switch screens)
- Since explicit intents shift from one Activity to another, there must be some way to send information to the activity that is being started. We will study that shortly.

Contexts…

- What is a context? We're glad you asked!
- In English, one definition of context is the surrounding environment, or background information, relating to some event
- In Android, a Context is an abstract class that provides *global information about an application's environment*
- A context can do many things, including interact with an app's resources, launch activities, send broadcasts, register broadcast receivers, getting internal file paths
- There are 2 types of contexts — one associated with the entire application (and there is only one of these, ever); and another associated with each component (activity, service, content provider and broadcast receiver).
- An Activity's context can do pretty much anything you need, so use it (except in situations where you don't have one)



```
java.lang.Object
   ↳android.content.Context
      ↳android.content.ContextWrapper
         ↳android.view.ContextThemeWrapper
            ↳android.app.Activity
               ↳android.support.v4.app.FragmentActivity
                  ↳android.support.v7.app.AppCompatActivity
```



	Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a Dialog	NO	YES	NO	NO	NO
Start an Activity	NO[1]	YES	NO[1]	NO[1]	NO[1]
Layout Inflation	NO[2]	YES	NO[2]	NO[2]	NO[2]
Start a Service	YES	YES	YES	YES	YES
Bind to a Service	YES	YES	YES	YES	NO
Send a Broadcast	YES	YES	YES	YES	YES
Register BroadcastReceiver	YES	YES	YES	YES	NO[3]
Load Resource Values	YES	YES	YES	YES	YES

Examples

- If the code creating the intent is in an Activity, it is already a context that we can use this to reference. If not, then we can get the context for the entire application.

```
Intent ini = new Intent(this, AnotherActivity.class);

Intent in2 = new Intent(getApplicationContext(),
      AnotherActivity.class);
```

Sending an Intent

- The intent not only is used as a trigger but is also sent to the target.

- **startActivity(Intent intent)**

- Intent can be either explicit or implicit.

Implicit Intents

- Implicit intents do not name an explicit component (class)
- Instead, use criteria (**Action**, **Data** & **Category**) and let the OS sort out which component (in *any app*, anywhere in the system) can respond.
- This gets a little complicated and so read over the next few slides, but don't obsess over them.
- We can specify all three in the constructor for the intent or use setAction(), setData(), and setCategory() with a default constructor.

Intent Action

- An **Action** is the general action to be performed - a String
- There are many of these, including:
    - ACTION_VIEW: the most common action, used to display data to the user
    - ACTION_EDIT: used to edit the data sent to the activity (or whatever component the intent is triggering)
    - ACTION_SEND: Deliver data to some other, unspecified activity. The user is usually expected to pick the activity, and as such the intent should be passed into createChooser().

Intent Data

- **Data** - the data which the action to operate on - a URI
  - e.g., content://contacts/people/1 -- a person in your contacts database
  - tel:6605551212 -- a telephone number to call
  - http://www.google.com – a well known web site.

Intent Category

- **Category** - additional information about the kind of component that should handle the intent
  - e.g., CATEGORY_LAUNCHER - used for the "main" activity of an app. This causes it to be listed in the application launcher (so the user can see it and tap on it).
  - CATEGORY_DEFAULT - the usual category for all other activities site.

Example

- Here is the code for an onClick method that will bring up an activity that will be ready to dial the given phone number. (If we try to use ACTION_CALL, we would need to specify that permission in the manifest.)
- The category will be the default.

```
public void dialAction(View v) {
   String uri = "tel:6605551212";
   Intent dial = new Intent(Intent.ACTION_DIAL);
   dial.setData(Uri.parse(uri));
   startActivity(dial);
}
```

Receiving Implicit Intents

- Each component has a manifest entry with an **intent-filter**. That intent-filter can also define an action, data and category.

Take another look at AndroidManifest.xml

- The OS examines the intent-filters to find the one(s) that match
- If more than one match, the user is given a choice
- Notes:
  - All components should be listed in the Android Manifest file.
  - However, if your component is self-contained and you do not wish anyone else to be able to trigger it, don't write an intent filter for it.
  - One of your components *must* have an intent filter, as Android needs to be able to send it an intent to launch it.

Example (Auto – Generated)

-
  ```
  <activity android:name=".MainActivity">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />

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

Example (Extended)

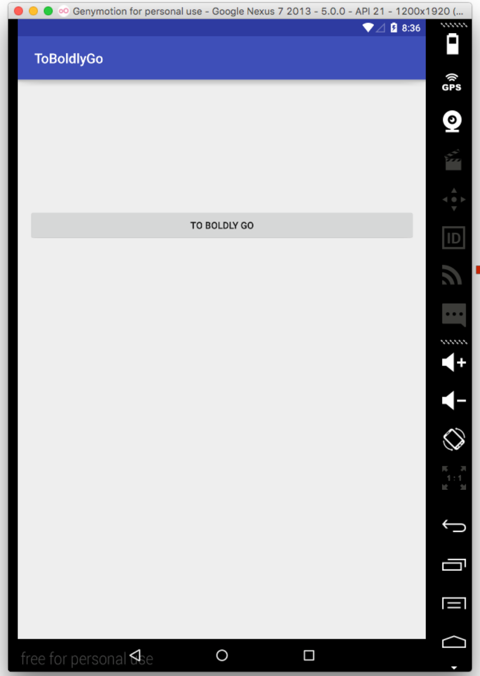
```
<activity android:name=“.ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

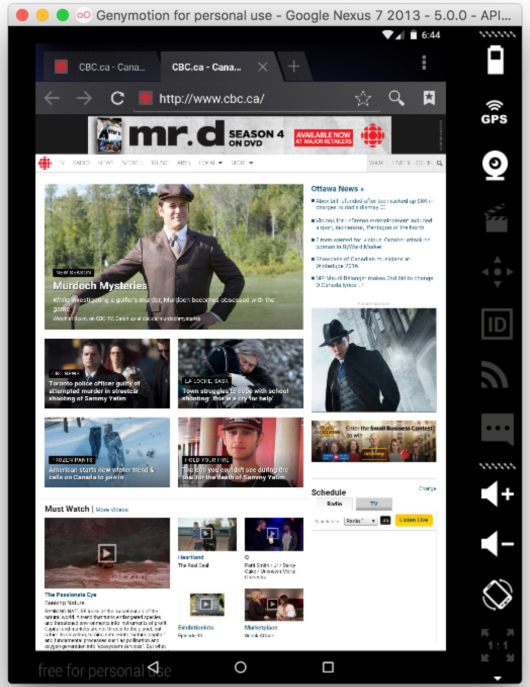
Must match all 3

Must match at least 1 action and 1 data and the 1 (and only) category

Another Example

```
public void engage(View v){
    Intent ini = new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.cbc.ca"));
    startActivity(ini);
    }
```


Genymotion for personal use - Google Nexus 7 2013 - 5.0.0 - API 21 - 1200x1920 (...
8:36
ToBoldlyGo
TO BOLDLY GO
free for personal use



Adding Information

- Since an intent is a message, we want it to be able to carry information.
- **Extras:** key-value pairs that provide additional info needed by the receiving component.
- There are a number of places where we want this flexibility in Android and a dictionary structure was selected as the solution. Unfortunately, there are subtle differences between each of the cases.

Adding Information

- **Sending the info:**
  - Intent putExtra(String key, *Primitive* value)
  - Intent putExtra(String key, Serializable value)
  - Intent putExtra(String key, String [] value)
  - Intent putExtras(Bundle bundle)
    - Add a bundle (another kind of dictionary) to the intent.
- **Receiving the info:**
  - *Primitive* get*Extra(String key, *Primitive* defaultValue)
    // * is any Primitive type, e.g., double, int, etc., as well as String
  - *Primitive* [] get*ArrayExtra(String key)
    // * is any Primitive, e.g., double, long, etc., as well as String
  - Serializable getExtra(String key, Serializable defaultValue)
-).

There are many overloaded putExtra() methods.

Adding Information

- Intent putExtra(String name, int value)
- int getIntExtra(String name, int defaultValue)

- Intent putExtra(String name, float value)
- float getFloatExtra(String name, float defaultValue)

- Intent putExtra(String name, int [] value)
- int [] getIntArrayExtra(String name)

Example:

```
/* On a map activity, the user clicks, getting
lat/long coordinates. Then, we want to, on another
activity, fetch details relating to that location
*/

Intent intent = new Intent(this, MapDetails.class);

intent.putExtra("Lat",latitude);

intent.putExtra("Long",longitude);

startActivity(intent);
```

```
/* Most likely in MapDetail's onCreate() method …

Intent intent = getIntent();
double latitude = intent.getDoubleExtra("Lat",0.0);
double long = intent.getDoubleExtra("Long",0.0);
```

Example:

```
Intent intent = new Intent(this, Mailer.class);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Explicit or Implicit?

```
// later, in the activity that was fired up …


Intent intent = getIntent();
String [] recipients =
        intent.getStringArrayExtra(Intent.EXTRA_EMAIL);
```

Examples:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, eg "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Explicit or Implicit?

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type
// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Explicit or Implicit?

Handling Start Errors:

It is poor form to let an App crash just because an implicit intent was unable to resolve. We have a couple options to prevent this from occurring.

```
// Option1: Use a try block
try { startActivity(someIntent)}
catch (ActivityNotFoundException e) { // Error handling code if needed
}


// Option2: Check to see if the package manager can find
// an activity that matches the intent
if (someIntent.resolveActivity(getPackageManager()) != null) {
      startActivity(sendIntent); }
```

Getting Back Results

- The underlying architecture described in the next two slides is still supported at all operating system levels as of Feb 8, 2022, but **startActivityForResul**t has been deprecated and you should move to Android Result API.

Adding Information

- You may wish to get a result *back* from an activity
- In that case, instead of **startActivity()**, invoke:
- **startActivityForResult(Intent intent, int requestCode)**
- When the called activity finishes, it must invoke
- **setResult(int resultCode) || setResult(int resultCode, Intent data)**
- **finish()**
  - finish() causes the activity to … finish! And then this is called:
  - **onActivityResult(int requestCode, int resultCode, Intent data)**

A Picture *May* Help:

```
onClick(View v){  //Or some other click
handler
  …
  ini.putExtra(“some key”, value);
  startActivityForResult(ini, 99);
```

99 is a request code that specifies where the request came from

```
onActivityResult(int req,
                 int ret,
                 Intent ini){
   if(req == 99)
      //Handle this request
}
```

```
onCreate(){
   Intent trigger = getIntent();
   value = trigger.getExtra(“some key”)
   // etc
}


onClick(View v) { //or another handler
   Intent trigger = getIntent();

   // … put results in trigger

  setResult(0, trigger)
  finish();
}
```

99 is automatic, 0 is the return code

trigger is a convenient intent

Getting Back Results (Post Androidx)

- The problem - When you transfer to another activity, there is a chance that the initiating activity has been destroyed. (E.G. you started the camera and it needs memory and the system responds by destroying activity instances.) The consequence is that we can not deal with any result until the instance that was destroyed is reinstated by the operating system.
- The solution - Decouple the handler code.

AndroidResult API

- Methods are provided to:
- Register for a result
    - Set up a call back that can be used to handle the result.
    - Gives us a launcher.
- Launch for a result - Launch the activity that has been registered.
- Handle the result - The activity result contract that is our result handler.

Register

- Method `registerForActivityResult` has:
- Contract - Specifies the input to the activity and the output we are getting back.
- Callback - What to do with the result that is returned
- Launcher - a returned object that allows us to initiate the activity

```
abstract @NonNull ActivityResultLauncher<I> <I,O>
  registerForActivityResult(
     @NonNull ActivityResultContract<I, O> contract,
     @NonNull ActivityResultCallback<O> callback )
```

Example - Implicit

- We are using an instance variable to hold the launcher.

```
// GetContent creates an ActivityResultLauncher<String>
// to allow you to pass
// in the mime type you'd like to allow the user to select
private ActivityResultLauncher<String> mGetContent;
```

Example (cont)

- Registration must be done before the activity is running - Can place this code in onCreate.
- Contract - getContent constructor - this will be implicit.
- Callback - ActivityResultCallBack with a result that is a Uri and the method is onActivityResult.
- Launcher - mGetContent

```
mGetContent
   = registerForActivityResult(
        new GetContent(),
        new ActivityResultCallback<Uri>() {
           @Override
           public void onActivityResult(Uri uri) {
              // Handle the returned Uri
           }
     });
```

Example (cont)

- Launch the activity.

```
// Pass in the mime type you'd like to allow the user to select
// as the input
mGetContent.launch(“image/*");
```

Example - Explicit

- We are using an instance variable to hold the launcher.

```
private ActivityResultLauncher<Intent> mStartForResult;
```

Example (cont)

- Registration code for onCreate().
- Contract - new instance created. This will be explicit
- Callback - ActivityResultCallBack with a result of type ActivityResult. This code assumes that the result is going to be an intent and will be in the data attribute of the result.
- Launcher - mStartForResult

```
mStartForResult
   = registerForActivityResult(
      new StartActivityForResult(),
      new ActivityResultCallback<ActivityResult>() {
         @Override
         public void onActivityResult(ActivityResult result) {
            if (result.getResultCode() == Activity.RESULT_OK) {
               Intent intent = result.getData();
               // Handle the Intent
            }
         }
      });
```

Example (cont)

- Returning the value.   This is the same as what we did before for explicit intents.
- We can do the same for implicit intents.

```
// We are returning the value from the child activity
// Using the built in result codes (or we can define our own)
setResult(Activity.RESULT_OK, anIntent);
finish();
```

Questions

- 1.What is the purpose of an intent?
- 2.What is the difference between an implicit and explicit intent? Which can be used to communicate among components in different apps?
- 3.Give the signature of the Intent constructor for explicit intents
- 4.Create an Intent, and use it to start a subclass of Activity called MyActivity.
- 5.What is the difference between startActivity() and startActivityForResult()?
- 6.What does finish() do?
- 7.Bundle the number 3.0E8 into an Intent (call it SPEED_OF_LIGHT), send it to an Activity called Rocket, and in the Rocket activity's onCreate() method, extract the 3.0E8 and print it to the console.
- 8.In an Activity, what does getIntent() return?
- 9.Describe the mechanism by which implicit intents are delivered

Questions 2

- 1.What are Actions, Categories, and Data used for?
- 2.Why are intent-filters embedded in <activity> elements?
- 3.Consider 2 activities, ActivityA and ActivityB. ActivityA is to launch ActivityB, and when ActivityB finishes, control should return to Activity A. Describe the mechanism, and the methods (including their signatures), involved in this.
- 4.Write the specific code to launch ActivityB from ActivityA, embed some information in an intent, and return it to ActivityA (as described in slides 14, 15, various doodles, and hopefully discovered by you during a homework assignment)

Resources

- https://developer.android.com/guide/components/ intents-common.html
- https://possiblemobile.com/2013/06/context/