**455-555 Week 4 Notes**

**Categories:**

Chapter 6 – More on Intents

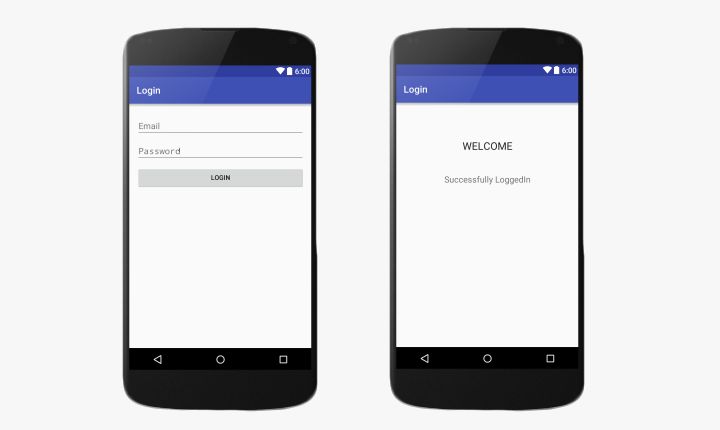
Lab 2 intro- Seek bar (see demo file), check box, view stub, list view and special adapters

Reading for Wednesday Session: Chapter 7 – JSON vs XML intro

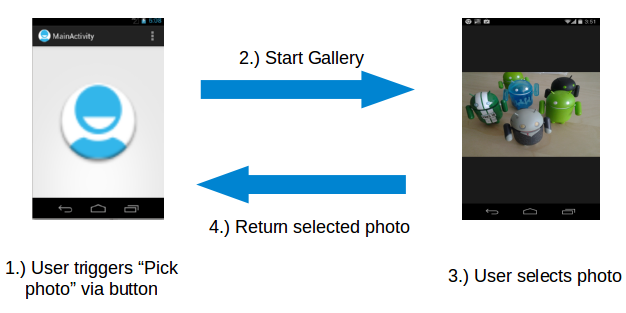
­­­­

**[ Chapter 6 – More on Intents ]**

*Intents* are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

Intents are objects of the **android.content.Intent** type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity

Explicit Intent



To create an intent for opening a photo in another activity in Android Studio using Java, you can use the **following code as an example:**

The following code snippet demonstrates how to create an intent to open a photo in another activity within or outside your own application even.

// Inside your current activity, when you want to open the photo in another activity

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

intent.putExtra("photoUri", yourPhotoUri); // Pass URI of photo to the new activity

startActivity(intent);

\*code for yourPhotoUri above can be as follows

// Replace this line with the actual URI of the photo you want to open

Uri yourPhotoUri = Uri.parse("content://media/external/images/media/12345");

Code on the receiving end…

// Inside the receiving activity, retrieve the photo URI from the intent

Intent intent = getIntent();

String photoUriString = intent.getStringExtra("photoUri");

Uri photoUri = Uri.parse(photoUriString);

// Use the MediaStore content provider to retrieve the photo

Bitmap photoBitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), photoUri);

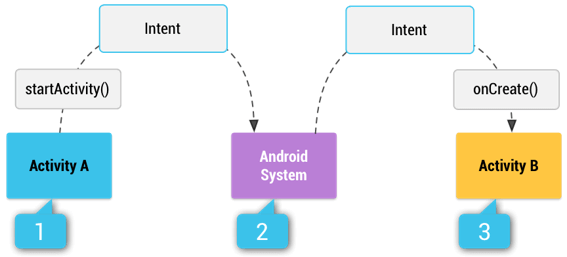
// Display the photo in an ImageView

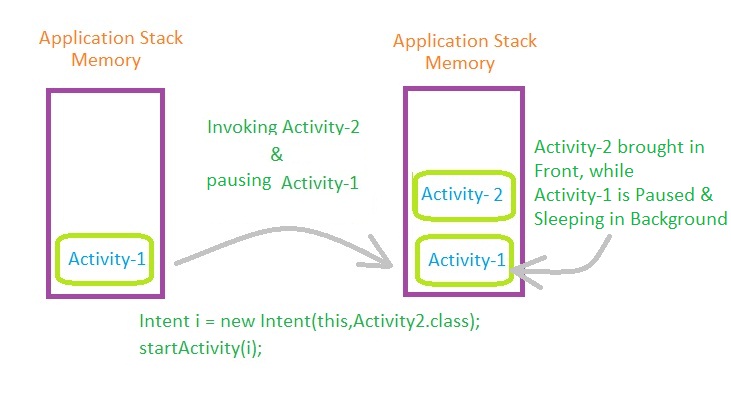
ImageView imageView = findViewById(R.id.imageView);

imageView.setImageBitmap(photoBitmap);

The **MediaStore** content provider provides a way for applications to access media files such as photos and videos stored on the device. By using this approach, the receiving application can securely and efficiently access photos pushed from different applications.

The provided code logic above and explanation are based on the Android [documentation](https://developer.android.com/topic/libraries/architecture/datastore) and best practices for accessing media files from shared storage using the **MediaStore** content provider





Explicit vs. Implicit Explained

Android supports *explicit* and *implicit* intents. An application can define the target component directly in the intent (explicit intent) or ask the Android system to evaluate registered components based on the intent data(implicit intents).

Explicit intents explicitly define the component which should be called by the Android system, by using the Java class as identifier. Explicit intents are typically used within an application as the classes in an application are controlled by the application developer. The following shows how to create an explicit intent and send it to the Android system to start an activity.

An intent can contain data via a *Bundle*. This data can be used by the receiving

component.

[**Explicit Intents**](https://developer.android.com/reference/android/content/Intent) **( Bundling data )**

**Activity 1 – send bundle**

*//create a Bundle object*Bundle extras = **new** Bundle();  
*//Adding key value pairs to* **this** *bundle  
//there are quite a lot data types you can store in a bundle*extras.putString(**"USER\_NAME"**,**"jhon Doe"**);  
extras.putInt(**"USER\_ID"**, 21);  
extras.putIntArray(**"USER\_SELECTIONS"**,**new int**[]{1, 2, 3, 4, 5});  
  
*//create and initialize an intent*Intent intent = **new** Intent(**this**, Main2Activity.**class**);  
*//attach the bundle to the Intent object*intent.putExtras(extras);  
*//finally start the activity*startActivity(intent);

**Activity 2 – receive bundle**

*//get the intent in the target activity*Intent intent = getIntent();  
*//get the attached bundle from the intent*Bundle extras = intent.getExtras();  
*//Extracting the stored data from the bundle*String user\_name = extras.getString(**"USER\_NAME"**);  
Integer user\_id = extras.getInt(**"USER\_ID"**);  
**int**[] array = extras.getIntArray(**"USER\_SELECTIONS"**);  
  
*// Display data*TextView textView = findViewById(R.id.***textView***);  
textView.setText(**""** + array[0]);

**NEWER API’s format!**

**SET DATA IN ONE ACTIVITY**

**Intent i = new Intent(packageContext:LoginActivity1.this, MainActivity.class);  
i.putExtra(name:"key", uName);  
i.putExtra(name:"key2", value:"success!");  
startActivity(i);**

**RETRIEVE DATA IN ANOTHER ACTIVITY**

**Intent intent = getIntent();**

**//get the attached extras from the intent  
//we should use the same key as we used to attach the data.  
String uName = intent.getStringExtra("key");**

**//if you have used any other type of data, you should use the  
//particular getExtra method to extract the data from Intent**

**txtUname.setText("Logged in as " + uName); //show resulting data**

Getting to the Web anyone- make use of Implicit Intents?

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

**Intent i = new Intent(Intent.ACTION\_VIEW, Uri.parse("http://www.iit.edu"));**

**startActivity(i);**

----

Bundle object passed (send/retrieve all data)

**Bundle extras = getIntent().getExtras();**

**if (extras == null) {**

**return;**

**}**

***// get data via the key***

**String value1 = extras.getString(Intent.SOME\_TEXT);**

**if (value1 != null) {**

***// do something with the data***

**}**

***// This is the callback for the started sub-activities***

**@Override**

**protected void onActivityResult(int requestCode, int resultCode, Intent data){**

**if (requestCode == SUB\_ACTIVITY\_CREATE\_USER && resultCode ==**

**Activity.RESULT\_OK) {**

**Bundle extras = data.getExtras();**

**if (extras != null) {**

**String name = extras.getString(User.USER\_NAME);**

**boolean gender = extras.getBoolean(User.USER\_GENDER);**

**user = new User(name, gender);**

**updateUserInterface();**

**}**

**}**

**}**

In the above call back method, a "sub-activity" refers to an activity that is started from another activity using startActivityForResult(). When the sub-activity finishes, it can return a result to the calling activity. The calling activity then receives the result in the onActivityResult() method. This mechanism is often used to delegate a specific task to another activity and receive a result back from it.

For example, if Activity A starts Activity B using startActivityForResult(), Activity B is considered a "sub-activity" of Activity A. When Activity B finishes and returns a result, the onActivityResult() method in Activity A is called to handle the returned result.

Also in the call back method, the **SUB\_ACTIVITY\_CREATE\_USER** is a request code(constant integer) that is used to identify the sub-activity when the result is returned to the calling activity. When the sub-activity finishes and returns a result, the **onActivityResult** method in the calling activity is triggered. The request code is used to determine *which* sub-activity is sending the result, especially when the calling activity may have started multiple sub-activities and needs to differentiate between them.

Adding in extra Activities with App start Launch capabilities!

\*Note, check your manifest for added intents!! Which intent will fire first?

\*\*Further note you can always check/choose which activities are or can be chosen to launch by going to your menu and choosing Run > Edit Configurations as shown next.

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

Example of creating a Launch Activity feature as an option follows. Merely include a new Activity by right clicking on your package and choosing New > Activity > Empty Views Activity. Then as an option check Launcher Acitvity as shown below.

A screenshot of a video

Description automatically generated

Result of an added Activity added to your manifest. Which intent will fire first?

**<activity  
 android:name=".LoginActivity1"  
 android:label="@string/title\_activity\_login1"  
 android:theme="@style/AppTheme.NoActionBar">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>**

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

**Lab 2** overview– continues with lab 1 and introduces a Checkbox, SeekBar, listview with an ArrayAdapter and a Viewstub.

**[ Using an ArrayAdapter with ListView ]**

**In Android development, any time we want to show a vertical list of scrollable items we will use a ListView which has data populated using an Adapter. The simplest adapter to use is called an ArrayAdapter because the adapter converts an ArrayList of objects into View items loaded into the ListView container.**

Diagram

Description automatically generated

The ArrayAdapter fits in between an ArrayList (data source) and the ListView (visual representation) and configures two aspects:

* Which array to use as the data source for the list
* How to convert any given item in the array into a corresponding View object

View controls

**ViewStub** - A ViewStub is an invisible, zero-sized View that can be used to lazily *inflate*

layout resources at runtime.

<https://developer.android.com/reference/android/view/ViewStub.html>

**Seekbar** - A SeekBar is an extension of ProgressBar that adds a draggable thumb.

<https://developer.android.com/reference/android/widget/SeekBar.html>

**Listview** -  is a view group that displays a list of scrollable items.

<https://developer.android.com/guide/topics/ui/layout/listview.html>

**Arrayadapter** widget - A concrete BaseAdapter that is backed by an array of arbitrary objects.

<https://developer.android.com/reference/android/widget/ArrayAdapter.html>

**Simple\_list\_item** - It tells the **listview** what layout to use for the individual rows.

<https://developer.android.com/reference/android/R.layout.html>

<http://stackoverflow.com/questions/6079344/what-is-android-r-layout-simple-list-item-1>

Event Listeners/Handlers

SeekBar.[OnSeekBarChangeListener](https://developer.android.com/reference/android/widget/SeekBar.OnSeekBarChangeListener) is a ***callback*** that notifies clients when the progress level has changed for the seek bar control by touch gestures, arrow keys/trackballs, etc. and has three possible abstract methods that need to be overriden for implementation. Two of the methods are onStartTrackingTouch and onProgressChanged. Can you name the third needed overriden method to implement?

abstract void **onProgressChanged(**SeekBar seekBar, int progress, boolean fromUser)

Need: Notification that the progress level has changed.

abstract void **onStartTrackingTouch(**SeekBar seekBar)

Need: Notification that the user has started a touch gesture.

abstract void **onStopTrackingTouch**(SeekBar seekBar)

Need: Notification that the user has finished a touch gesture.

**Bonus: Adding in an Image for image button for lab 2 logic follows…**

Grab icon image from Google images

Save image file to the drawables folder

Setting up a **back** button follows…

<ImageButton

*<!— use dp’s for layout height / width -->*

android:id="@+id/imageButton"  
 android:layout\_width="52dp"  
 android:layout\_height="46dp"  
 **android:onClick="onClick"**  
 **android:src=**"@drawable/back"

/>

Now do the action!

**public void** onClick(View view) {

TextView textView2 = findViewById(R.id.***textview2***);

**checkBox**.setVisibility(View.***VISIBLE***);  
 **seekBar**.setVisibility(View.***VISIBLE***);  
 **stub**.setVisibility(View.***GONE***);  
 textView2.setVisibility(View.***GONE***);  
 **textView**.setVisibility(View.***VISIBLE***);  
  
 **img**.setVisibility(View.***GONE***);

*//also uncheck checkbox!!*

}

Reviews for Week 4 now loaded!

Extra References: [Recycleview](https://www.akshayrana.in/2020/06/recyclerview-and-cardview-example-android.html) and [Cardview](https://www.akshayrana.in/2020/06/recyclerview-and-cardview-example-android.html)