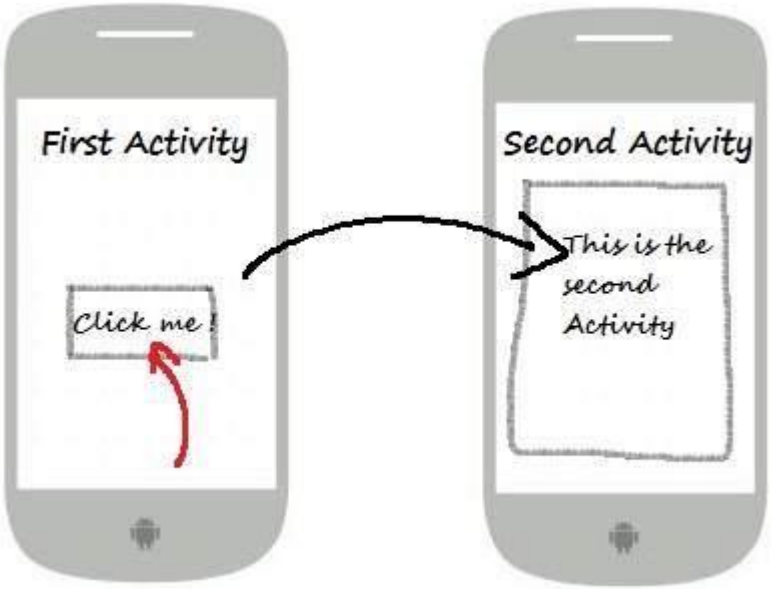
Mobile Development
Lecture 8: Intents and Animation

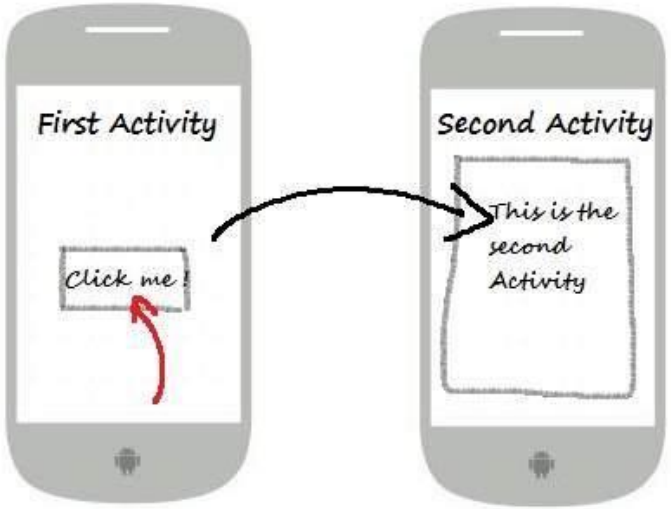
Mahmoud El-Gayyar
elgayyar@ci.suez.edu.eg
Elgayyar.weebly.com

1. Multiple Activities Intents



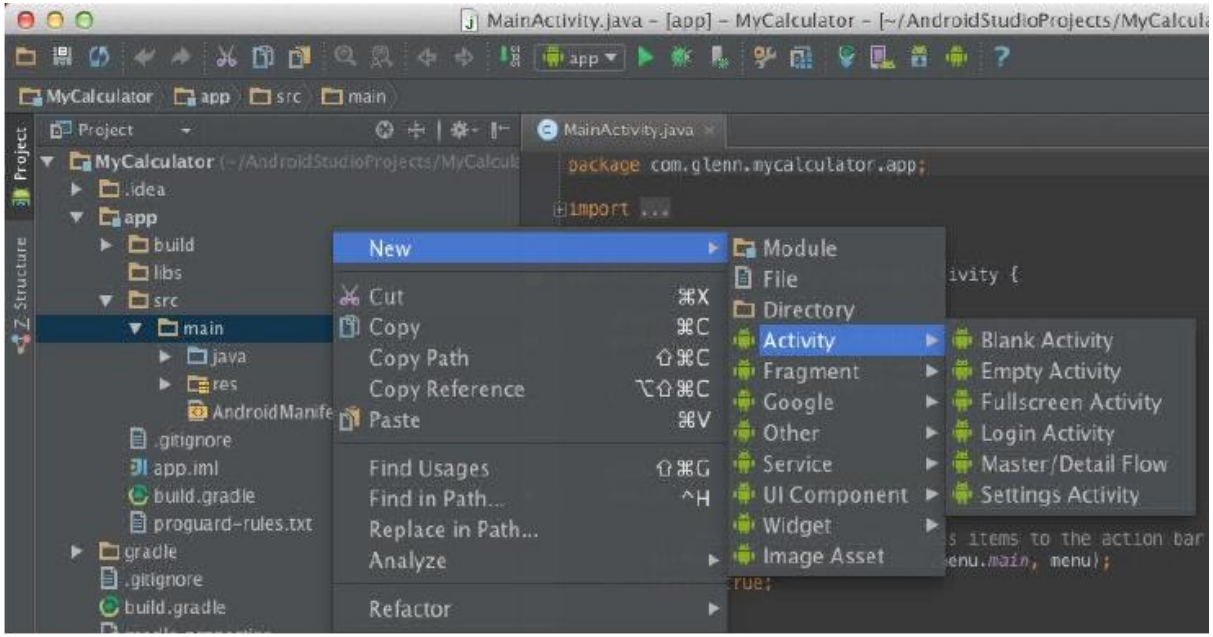
Multiple Activities

- *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.



Adding an Activity

- *in Android Studio, right click "app" at left: New -> Activity*
  - creates a **new .XML** file in res/layouts
  - creates a **new .java** class in src/java
  - adds information to **AndroidManifest.xml** about the activity
    - *(without this information, the app will not allow the activity)*



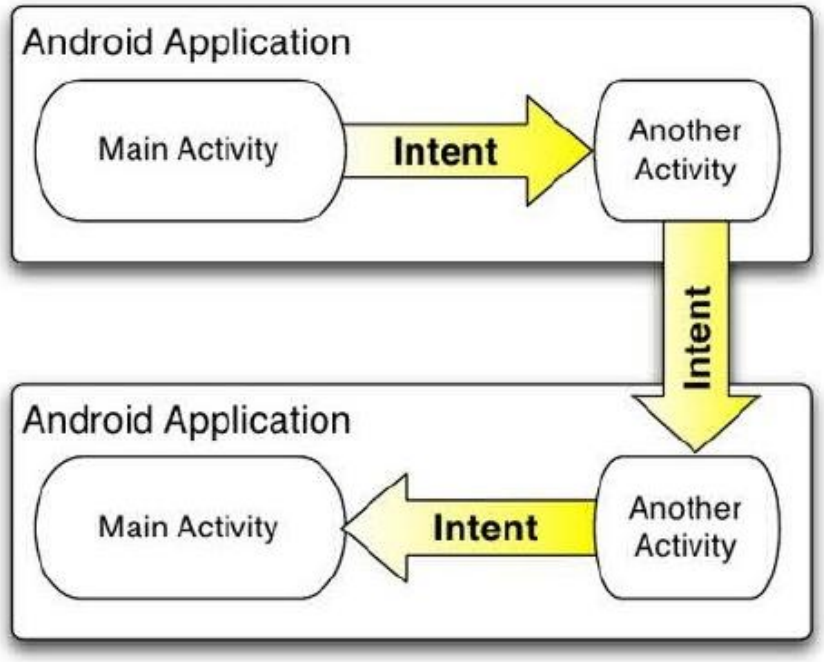
Activities in Manifest

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

```
<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:label="@string/title_activity_second"
                  android:parentActivityName=".SecondActivity" >
            <meta-data android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.myusername.myapplication.MainActivity" />
        </activity>
    </application>
</manifest>
```

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



Creating an Intent

- *To launch another activity (usually in response to an event), create an Intent object and call startActivity with it:*

```
Intent intent = new Intent(this, ActivityName.class);
startActivity(intent);
```

- *If you need to pass any parameters or data to the second activity, call putExtra on the intent.*
  - It stores "extra" data as key/value pairs, not unlike a Map.

```
Intent intent = new Intent(this, ActivityName.class);
intent.putExtra("name", value);
intent.putExtra("name", value);
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 act.*
  - You can access the Intent by calling **getIntent.**
  - The Intent has methods like getExtra, getIntExtra, getStringExtra, etc. to extract any data that was stored inside the intent.

```
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");
          ...
     }
}
```

Waiting for a Result

- *If calling activity wants to wait for a result from called activity:*
  - Call **startActivityForResult** rather than startActivity.
    - *startActivityForResult requires you to pass a unique ID number to represent the action being performed.*
    - *By convention, you declare a final int constant with a value of your choice. The call to startActivityForResult will not wait; it will return immediately.*
  - Write an **onActivityResult** method that will be called when the second activity is done.
    - *Check for your unique ID as was passed to startActivityForResult.*
    - *If you see your unique ID, you can ask the intent for any extra data.*
  - Modify the called activity to send a result back
    - *Use its **setResult** and finish methods to end the called activity.*

Sending Back a Result

- *In the second activity that was invoked, send data back:*
  - Need to create an Intent to go back.
  - Store any extra data in that intent; call setResult and finish.

```
public class SecondActivity extends Activity {
    ...
    public void myOnClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("name", value);
        setResult(RESULT_OK, intent);
        finish();      // calls onDestroy
    }
}
```

Grabbing the Result

```
    private static final int REQ_CODE = 123;    // MUST be 0-65535


    public void myOnClick(View view) {
        Intent intent = getIntent(this, SecondActivity.class);
        startActivityForResult(intent, REQ_CODE);
    }

    protected void onActivityResult(int requestCode,
            int resultCode, Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);
        if (requestCode == REQ_CODE) {
            // came back from SecondActivity
            String data = intent.getStringExtra("name");
            Toast.makeText(this, "Got back: " + data,
                           Toast.LENGTH_SHORT).show();
        }
    }
}
```

Implicit Intent

- ***implicit intent:*** *One that launches another app, without naming that specific app, to handle a given type of request or action.*
  - examples: invoke default browser; load music player to play a song

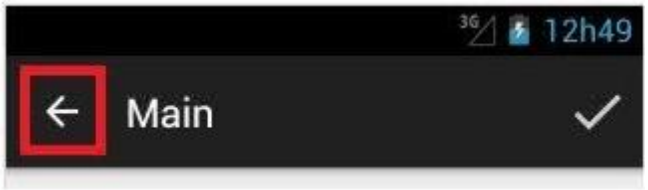
```
// make a phone call
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);

// go to a web page in the default browser
Uri webpage = Uri.parse("http://www.stanford.edu/");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);

// open a map pointing at a given latitude/longitude (z=zoom)
Uri location = Uri.parse("geo:37.422219,-122.08364?z=14");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

Activities and Action Bar

- ***action bar:*** *A top-level menu of actions in an activity.*
  - identifies current activity/app to user
- *If your activity is specified to have a "parent" activity on creation and in AndroidManifest.xml, you will have a "back" button to return to the calling activity.*



2. Animation

Types of Animation Frameworks

- ***Property Animations*** *- The most powerful and flexible animation system introduced in Android 3.0.*
- ***View Animations*** *- Slower and less flexible; deprecated since property animations were introduced*
- ***Transition Animations*** *- For Android 4.4 devices and above, the Transitions API framework enables layout changes within an activity. Using the design support library.*

Types of Animations

- ***Property Animations*** *- This is the animation of any property between two values. (rotating an image or fading out a button)*
- ***Activity Transitions*** *- Animates the transition as an Activity enters the screen*
- ***Fragment Transitions*** *- Animates the transition as a fragment enters or exits the screen*
- ***Layout Animations*** *- This allows us to enable animations on any layout container or other ViewGroup such as LinearLayout, RelativeLayout, or ListView. Using the Transitions API (for Android 4.4 devices and above), the animations to the view changes can be specified. For lower versions, layout animations can still be enabled, but there is no way to dictate how the transitions occur.*
- ***Drawable Animations*** *- Used to animate by displaying drawables in quick succession*

Property Animations

Property	Description
alpha	Fade in or out
rotation, rotationX, rotationY	Spin or flip
scaleX, scaleY	Grow or shrink
x, y, z	Position
translationX, translationY, translationZ (API 21+)	Offset from Position

Fading Animation (Property)

- *Can apply on different widgets*
  - Button, Images, …

```
component.animate().alpha(0f).setDuration(2000);
```

  - Time in milliseconds
  - Alpha between 0 – 1
  - Use setAlpha() method to change alpha before animation

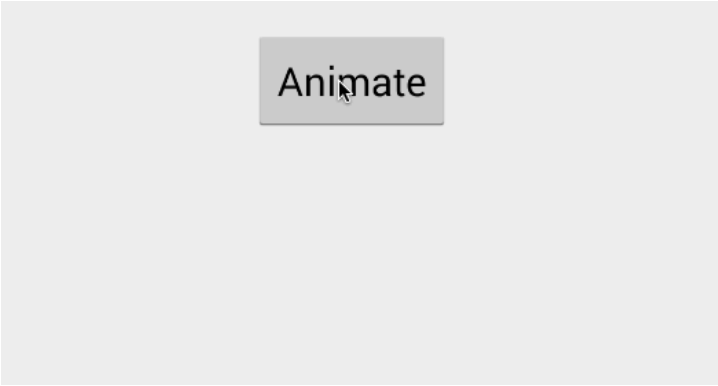
- *Using ObjectAnimator*

```
ObjectAnimator fadeAnim = ObjectAnimator.ofFloat(tvLabel, "alpha", 0.2f);
fadeAnim.start();
ObjectAnimator fadeAltAnim = ObjectAnimator.ofFloat(image, View.ALPHA, 0, 1);
fadeAltAnim.start();
```

Setting Duration or Repeat or Interpolation

```
ObjectAnimator scaleAnim = ObjectAnimator.ofFloat(tvLabel, "scaleX", 1.0f, 2.0f);
scaleAnim.setDuration(3000);
scaleAnim.setRepeatCount(ValueAnimator.INFINITE);
scaleAnim.setRepeatMode(ValueAnimator.REVERSE);              // or restart
scaleAnim.start();
```

```
ObjectAnimator moveAnim = ObjectAnimator.ofFloat(v, "Y", 1000);
moveAnim.setDuration(2000);
moveAnim.setInterpolator(new BounceInterpolator());
 moveAnim.start();
```



Common Interpolators

Name	Description
AccelerateInterpolator	Rate of change starts out slowly and and then accelerates
BounceInterpolator	Change bounces right at the end
DecelerateInterpolator	Rate of change starts out quickly and and then decelerates
LinearInterpolator	Rate of change is constant throughout

Others

Listening to the Animation Lifecycle

- *We can add an **AnimatorListenerAdapter** to manage events during the animation lifecycle such as onAnimationStart or onAnimationEnd:*

```
ObjectAnimator anim = ObjectAnimator.ofFloat(v, "alpha", 0.2f);
anim.addListener(new AnimatorListenerAdapter() {
   @Override
   public void onAnimationEnd(Animator animation) {
    Toast.makeText(MainActivity.this, "End!", Toast.LENGTH_SHORT).show();
    }
});

anim.start();
```

Multiple Animations

- *We can play multiple ObjectAnimator objects together concurrently with the AnimatorSet with:*

```
AnimatorSet set = new AnimatorSet();
set.playTogether(
    ObjectAnimator.ofFloat(tvLabel, "scaleX", 1.0f, 2.0f)
        .setDuration(2000),
    ObjectAnimator.ofFloat(tvLabel, "scaleY", 1.0f, 2.0f)
        .setDuration(2000),
    ObjectAnimator.ofObject(tvLabel, "backgroundColor", new ArgbEvaluator(),
          /*Red*/0xFFFF8080, /*Blue*/0xFF8080FF)
        .setDuration(2000)
);
set.start();
```

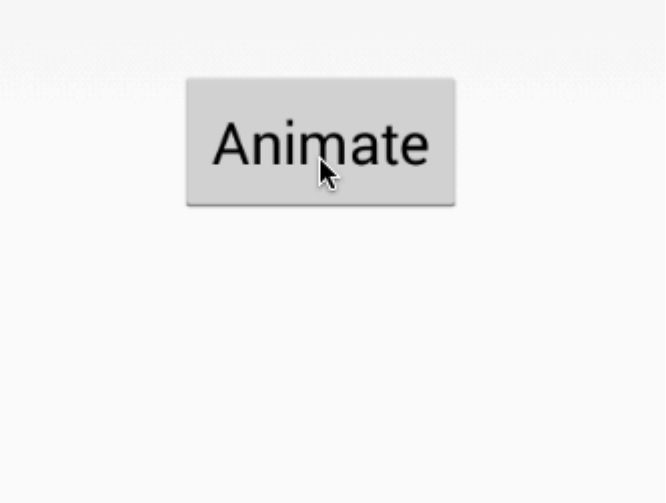
Using XML

- *We can also use property animations from XML.*
  - create an XML file that describes the object property animation we want to run. F
  - or example, if we wanted to animate a fade out for a button, we could add this file to **res/animator/fade_out.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
xmlns:android="http://schemas.android.com/apk/res/android"
android:propertyName="alpha"
android:duration="1000"
android:valueTo="0" />
```

```
Animator anim = AnimatorInflater.loadAnimator(this,
                                              R.animator.fade_out);
anim.setTarget(btnExample);
anim.start();
```

Challenge

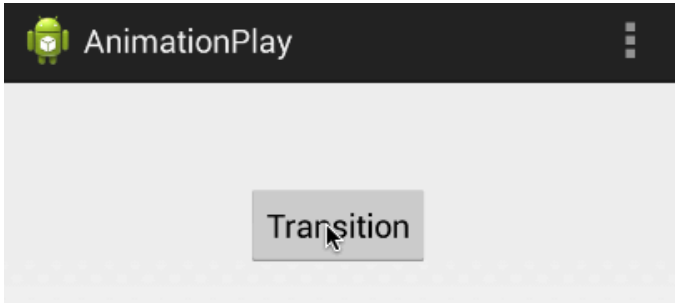


Activity Transitions

- *Animations and transitions for Activities as they become visible on screen*
  - After executing an intent with startActivity, you call the overridePendingTransition

```
Intent i = new Intent(MainActivity.this, SecondActivity.class);
startActivity(i);
overridePendingTransition(R.anim.right_in, R.anim.left_out);
```

  - The first parameter is the "enter" animation for the launched activity and the second is the "exit" animation for the current activity.
  - You can call on Activity exist also (e.g. after finish( ) method)



Layout Animation - 1

- *A particular animation can be specified when the layout first appears on screen. This can be done by using the **android:layoutAnimation** property to specify an animation to execute.*
  - First, let's define an animation we'd like to use when the views in the layout appear on the screen in res/anim/slide_right.xml which defines sliding in right from outside the screen:

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/accelerate_interpolator">
    <translate android:fromXDelta="-100%p" android:toXDelta="0"
            android:duration="1000" />
</set>
```

Layout Animation - 2

- *and now we need to create a special layoutAnimation which references that animation:*

```
<?xml version="1.0" encoding="utf-8"?>
<layoutAnimation
xmlns:android="http://schemas.android.com/apk/res/android"
          android:animation="@anim/slide_right"
/>
```

- *and now we need to apply this layoutAnimation to our Layout or ViewGroup:*

```
<LinearLayout
    ...
    android:layoutAnimation="@anim/layout_slide_in_from_right" />
```

Tutorial 1 Tutorial 2

Animated Images

- *In certain cases, we want to be able to display animated images such as an simple animated **GIF**. The underlying class for making this happen is called AnimationDrawable which is an XML format for describing a sequence of images to display.*
  - One of the simplest ways to display an animated gif is to use a third-party library.
- *First, let's add Glide to our app/build.gradle file:*

```
dependencies {
    ...
    compile 'com.github.bumptech.glide:glide:3.6.0'
}
```

- *Next, setup an ImageView in the the activity (@+id/ivGif)*
- *For local GIFs, be sure to put an animated gif into the **res/raw***

Load GIF with Glide

```
protected void onCreate(Bundle savedInstanceState) {
        …
        // Find the ImageView to display the GIF
        ImageView ivGif = (ImageView) findViewById(R.id.ivGif);
        // Display the GIF (from raw resource) into the ImageView
        Glide.with(this).load(R.raw.my_gif).asGif().into(imageView);
        // OR even download from the network
        Glide.with(this).load("https://i.imgur.com/l9lffwf.gif").asGif().into(imageView);
    }
}
```