Playing sound effects (link)

- Find sound files such as .WAV, .MP3
- put sound files in project folder **app/src/main/res/raw**
- in Java code, refer to audio file as R.raw.***filename***
  - (don't include the extension; R.raw.foo for foo.mp3)
  - use simple file names with only letters and numbers

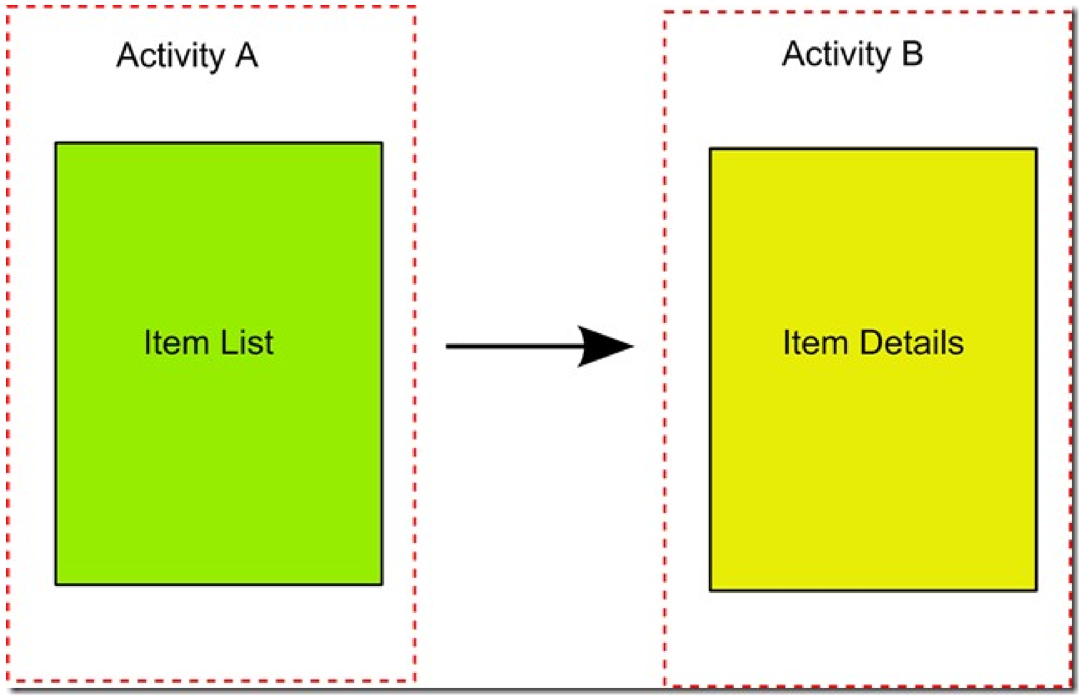
- Load and play clips using Android's MediaPlayer class

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.filename);
mp.start();
```

  - other methods: **stop**, **pause**, isLooping, **isPlaying**, getCurrentPosition, **release**, seekTo, setDataSource, setLooping

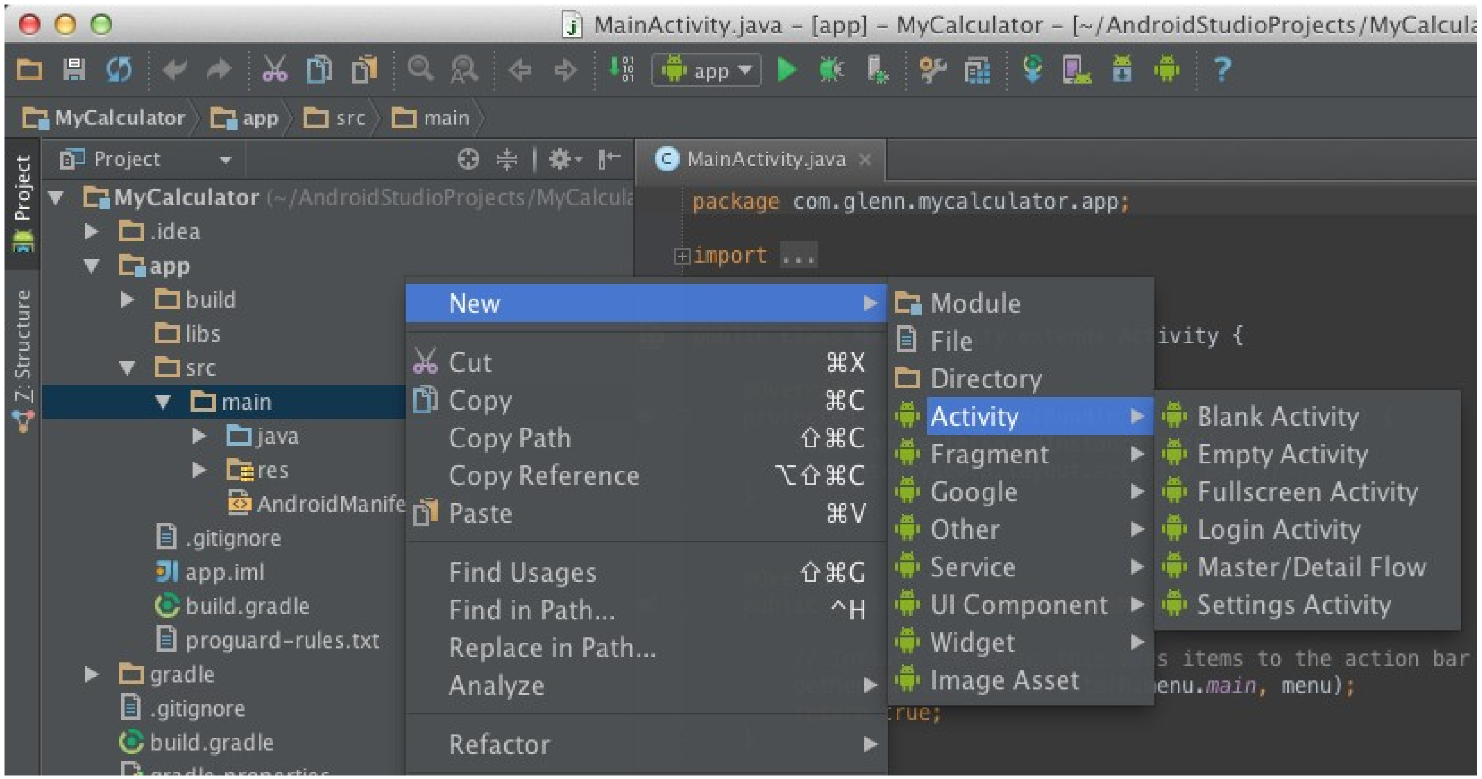
Multiple Activities

- 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)



```
package com.glenn.mycalculator.app;

import ...
```

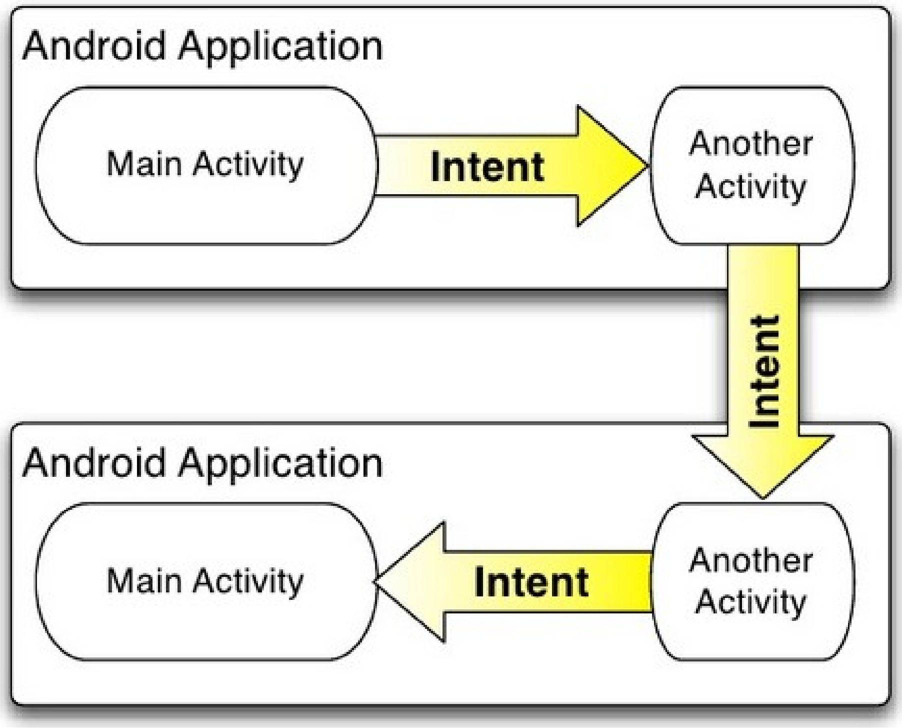
Activities in Manifest

- Every activity has an entry in project's **AndroidManifest.xm**l, 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 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.

```
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

```
public class FirstActivity extends Activity {
    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();
        }
    }
}
```