



Mobile Application Development

COMP-304

Fall 2018



Anatomy and Life Cycle of Android Applications

Objectives:

- ☐ Explain Android **activities**, **intents**, and **application life cycle**.
- ☐ Use **intents** to **call built-in applications** and **pass information** to other activities.

NOTE: AppCompatActivity class vs Activity class, is used for backwards compatibility



introduced in Android N, fragments
- miniature activities that can be grouped together to form an activity

Activities

intents
- glue that enables activities from diff apps to work together

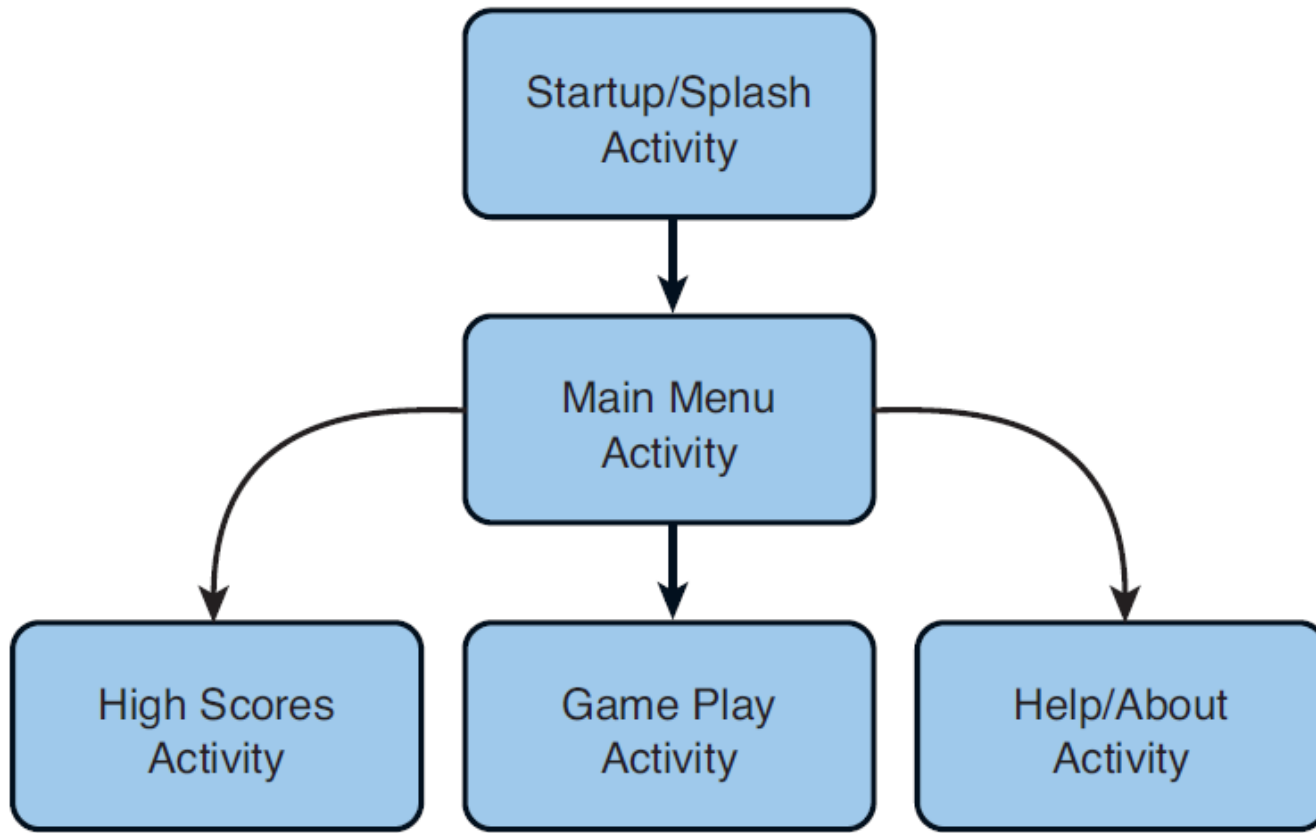
- ❑ **Activity**: is an application **component that provides a screen with which users can interact in order to do something**, such as dial the phone, take a photo, send an email, or view a map.
 - Each activity is given a window in which to draw its user interface.
- ❑ To create an activity, you must *create a subclass of Activity*:

```
public class Activity101Activity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```



Performing Application Tasks with Activities

- ❑ The Android **Activity** class (`android.app.Activity`) is core to any Android application:





The Lifecycle of an Android Activity

- ❑ Android applications can be **multi-process**, and the Android OS allows **multiple applications** to run



I am the top Activity.
User can see and interact with me!



I am the second Activity in the stack.
If the user hits Back or the top Activity is destroyed,
the user can see and interact with me again!



I am an Activity in the middle of the stack.
Users cannot see and interact with me until everyone
above me is destroyed.



I am an Activity at the bottom of the stack.
If those Activities above me use too many resources,
I will be destroyed!



The Lifecycle of an Android Activity

- ❑ When an activity transitions into and out of the different states, it is notified through various *callback or life cycle methods*.
- ❑ All the life cycle methods are hooks that you can override to do appropriate work when the state of your activity changes:

```
public class MyActivity extends Activity
```

```
{
```

```
    protected void onCreate(Bundle savedInstanceState);
```

```
    protected void onStart();
```

```
    protected void onRestart();
```

```
    protected void onResume();
```

```
    protected void onPause();
```

```
    protected void onStop();
```

```
    protected void onDestroy();
```

```
}
```

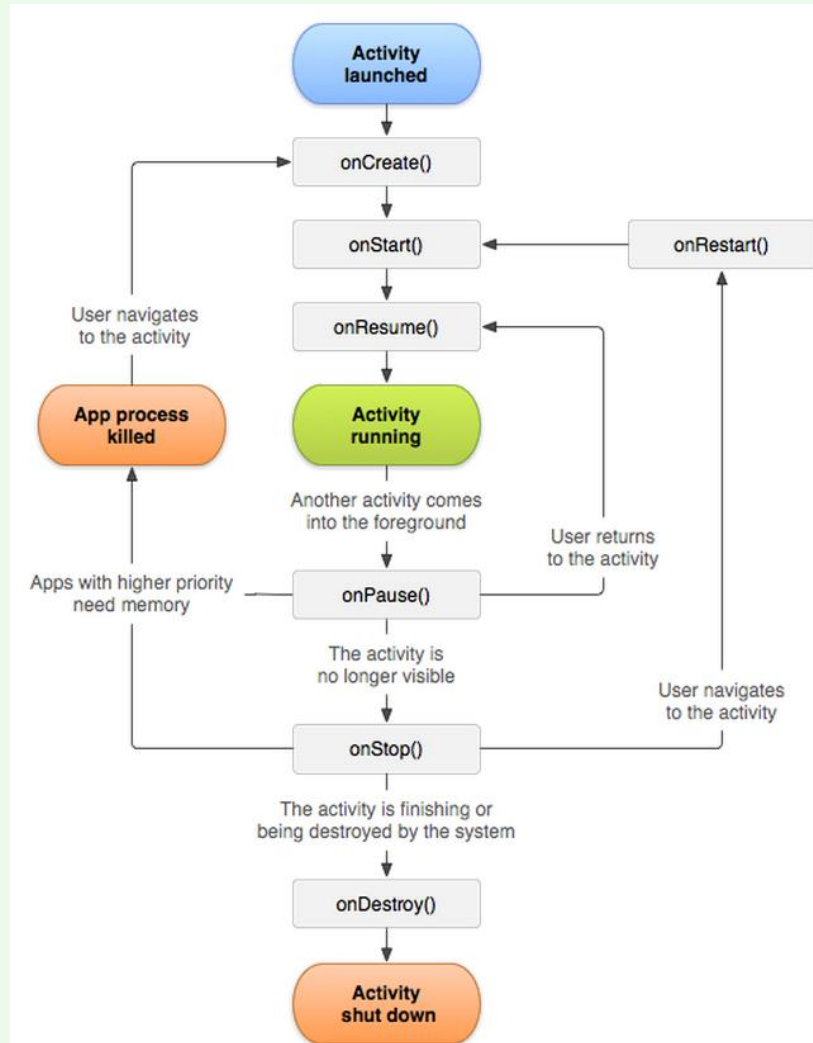
activities are destroyed when you click the Back button; w/e state the activity was in will be lost

onStart -> onResume called regardless whether newly created activity or restored from background



The Lifecycle of an Android Activity

Loops and the paths an activity might take between states:





Life Cycle Methods

- ❑ **onCreate()** - Called when the activity is first created
 - perform basic application startup logic that should happen only once for the entire life of the activity
 - **layout** and **data binding**, in the **onCreate()** method - this includes calls to the **setContentView()** method
- ❑ **onStart()** - makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.
 - this method is where the app initializes the code that maintains the UI.
 - It will be followed by **onResume()**
- ❑ **onResume()** - called when the activity will start interacting with the user, activity becomes the **foreground process**
 - **Initialize and retrieve** Activity data in **onResume()**
 - **start audio, video, and animations** here



Life Cycle Methods

- ❑ **onPause()** - to pause or adjust operations that should not continue
 - the user is leaving your activity, the activity is no longer in the foreground
 - **release** system resources here
 - **stop any audio, video**, and animations it started in the `onResume()` method
- ❑ **onStop()** - a newly launched activity covers the entire screen, or when the activity has finished running, and is about to be terminated
 - Under low-memory conditions, the Android OS can kill the process for any Activity that has been **paused**, **stopped**, or **destroyed**
- ❑ The Activity **state** is saved into a **Bundle** object, assuming the Activity implements and uses **onSaveInstanceState()** for custom data
 - some View data is automatically saved
- ❑ **onDestroy()** - is called before the activity is destroyed
 - clean up anything it needs to before the Activity is destroyed.
i.e. free up resources; even if activity is killed application is still running in memory



Saving activity state

- ❑ When an activity is paused or stopped, the state of the activity is retained
- ❑ When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the Bundle that the system passes to your activity.
- ❑ Both the `onCreate()` and `onRestoreInstanceState()` callback methods receive the same Bundle that contains the instance state information.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState); // Always call the superclass first  
  
    // Check whether we're recreating a previously destroyed instance  
    if (savedInstanceState != null) {  
        // Restore value of members from saved state  
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);  
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);  
    } else {  
        // Probably initialize members with default values for a new instance  
    }  
}
```



Activity101 example

```
public class Activity101Activity extends Activity {  
    String tag = "Lifecycle";  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Log.d(tag, "In the onCreate() event");  
    }  
    public void onStart()  
    {  
        super.onStart();  
        Log.d(tag, "In the onStart() event");  
    }  
}
```



Activity101 example

```
public void onRestart()
{
super.onRestart();
Log.d(tag, "In the onRestart() event");
}
public void onResume()
{
super.onResume();
Log.d(tag, "In the onResume() event");
}
public void onPause()
{
super.onPause();
Log.d(tag, "In the onPause() event");
}
```



Activity101 example

```
public void onStop()  
{  
  super.onStop();  
  Log.d(tag, "In the onStop() event");  
}  
public void onDestroy()  
{  
  super.onDestroy();  
  Log.d(tag, "In the onDestroy() event");  
}  
}
```



Activity101 example

Activity101 - [C:\Classes\COMP304\class_examples_androidstudio\Activity101] - [app] - ..\app\src\main\java\net\learn2develop\Activity101\Activity101Activity.java - Android Studio 1.3.2

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Activity101 | app | src | main | java | net | learn2develop | Activity101 | Activity101Activity

Android

- app
 - manifests
 - java
 - net.learn2develop.Activity101
 - Activity101Activity
 - res
 - resources
 - Gradle Scripts
 - build.gradle (Project: Activity101)
 - build.gradle (Module: app)

Build Variants

Test Artifact: Android Instrumentation Tests

Module	Build Variant
app	debug

```
public class Activity101Activity extends Activity {
    String tag = "Lifecycle";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---hides the title bar---
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }

    public void onStart() {
        super.onStart();
        Log.d(tag, "In the onStart() event");
    }

    public void onRestart() {
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
    }

    public void onResume() {

```

Android

Emulator Nexus_5_API_23 Android 6.0 (API 23) net.learn2develop.Activity101 (2003)

Logcat ADB logs Memory CPU

Log level: Verbose

Show only selected application

09-13 13:40:47.240 2003-2003/net.learn2develop.Activity101 I/art: VM created with code_cache_capacity=4mb compile_tnresnoio=1000

09-13 13:46:47.271 2003-2003/net.learn2develop.Activity101 W/System: ClassLoader referenced unknown path: /data/app/net.learn2develop.Activity101-1/11b/x86

09-13 13:46:47.275 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onCreate() event

09-13 13:46:47.276 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onStart() event

09-13 13:46:47.276 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onResume() event

09-13 13:46:47.277 2003-2003/net.learn2develop.Activity101 D/OpenGLES: Use EGL_SWAP_BEHAVIOR_PRESERVED: true

09-13 13:46:47.281 2003-2003/net.learn2develop.Activity101 D/ HostConnection: get() New Host Connection established 0xad7b20d0, tid 2003

09-13 13:46:47.303 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onPause() event

09-13 13:46:47.330 2003-2018/net.learn2develop.Activity101 D/ HostConnection: get() New Host Connection established 0xad7b21b0, tid 2018

09-13 13:46:47.338 2003-2018/net.learn2develop.Activity101 I/OpenGLES: Initialized EGL, version 1.4

09-13 13:46:47.360 2003-2018/net.learn2develop.Activity101 W/EGL_emulation: eglSurfaceAttrib not implemented

09-13 13:46:47.360 2003-2018/net.learn2develop.Activity101 W/OpenGLES: Failed to set EGL_SWAP_BEHAVIOR on surface 0xabf2b240, error=EGL_SUCCESS

09-13 13:46:47.362 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onStop() event

09-13 13:47:09.832 2003-2010/net.learn2develop.Activity101 W/art: Suspending all threads took: 5.916ms

09-13 13:54:04.153 2003-2010/net.learn2develop.Activity101 W/art: Suspending all threads took: 10.816ms

09-13 13:56:46.134 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onRestart() event

09-13 13:56:46.139 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onStart() event

09-13 13:56:46.139 2003-2003/net.learn2develop.Activity101 D/Lifecycle: In the onResume() event

09-13 13:59:34.638 2003-2010/net.learn2develop.Activity101 W/art: Suspending all threads took: 11.503ms

Terminal Messages Android Run TODO

Event Log Gradle Console

Session 'app' running (14 minutes ago)

20:00 15% LITE 9% Contrast: app contrast



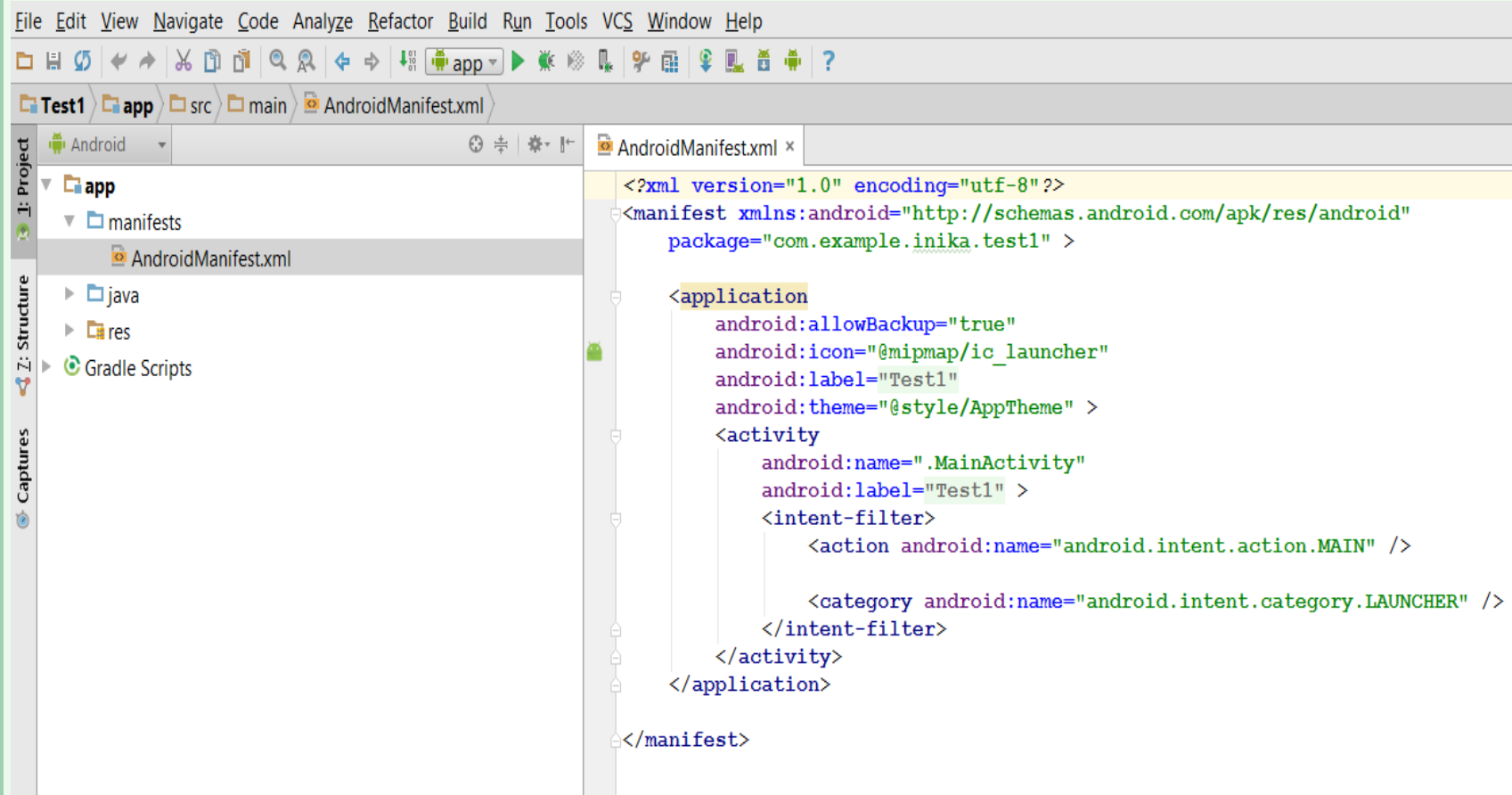
Defining Your Application Using the Android Manifest File

- ❑ Every app project must have an `AndroidManifest.xml` file (with precisely that name) at the root of the project source set.
- ❑ This file contains important information about the application's identity:
 - **The app's package name** - replaced with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play
 - The **components of the app** (activities, services, broadcast receivers, and content providers)
- ❑ The **permissions** that the app needs in order to access protected parts of the system or other apps.
- ❑ The **hardware and software features** the app requires, which affects which devices can install the app from Google Play.



Editing the Android Manifest File

- ❑ The **package name** must be defined within the Android manifest file within the `<manifest>` tag using the package attribute:





Setting the Application Name and Icon

- ❑ Set the application icon to a **drawable resource** provided with the application package and the application label to a **string resource**:

```
<application android:icon="@drawable/icon"  
    android:label="@string/app_name">
```

- ❑ set **optional application settings** as attributes in the `<application>` tag, such as the application description (`android:description`) and the setting to enable the application for debugging on the device (`android:debuggable="true"`).



Enforcing Application System Requirements

- ❑ Application system requirements that developers can configure through the Android manifest file include:
 - The Android **SDK versions** supported by the application
 - The Android **platform features** used by the application
 - The Android **hardware configurations** required by the application
 - The **screen sizes and pixel densities** supported by the application
 - Any **external libraries** that the application links to
- ❑ *defaultConfig* section in build.gradle overrides some manifest configurations.



Setting Android SDKs

- ❑ Android manifest file using the `<uses-sdk>` tag.
- ❑ This tag has three important attributes:
 - **The minSdkVersion attribute: specifies the lowest API level that the application supports:**

```
<uses-sdk android:minSdkVersion="19" />
```
 - **The targetSdkVersion attribute: specifies the optimum API level that the application supports:**

```
<uses-sdk android:minSdkVersion="19"  
          android:targetSdkVersion="23"
```
 - **The maxSdkVersion attribute: specifies the highest API level that the application supports:**
 - It restricts forward-compatibility of your application
 - Rarely used



Specifying Supported Screen Sizes

- ❑ The Android platform categorizes **screen types in terms of sizes** (small, normal, and large) and **pixel density (low, medium, and high)**.
- ❑ The `<supports-screens>` tag can be used to specify which Android **types of screens the application supports**.
- ❑ For example, if the application supports QVGA screens (small) and HVGA screens (normal) regardless of pixel density, the `<supports-screen>` tag is configured as follows:

```
<supports-screens android:smallScreens="true"  
android:normalScreens="true"  
android:largeScreens="false"  
android:anyDensity="true"/>
```



Registering Activities

- ❑ The following XML code defines an Activity class called ActivityA:

```
<activity android:name=".ActivityA">
```

- ❑ You may specify the complete class name:

```
<activity android:name="com.example.android.lifecycle.ActivityA" />
```

- ❑ You can also enforce scope of the activity class by using the dot as a prefix in the Activity name:

```
<activity android:name=".ActivityB" />
```



Designating a Primary Entry Point Activity

- ❑ **Intent Filters** – used by app components to describe a capability of that component.
 - The following tag of XML configures the Activity class called ActivityA as the **primary launching point** of the application:

```
<activity android:name=".ActivityA"
    android:launchMode="singleTask">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```



Registering Permissions

- ❑ Android applications have **no permissions by default**
- ❑ **Must be explicitly registered** within the Android manifest file
- ❑ This example defines a permission using the `<uses-permission>` tag to gain access to the built-in camera:

`<uses-permission android:name="android.permission.CAMERA" />`



Managing Application Resources

- ❑ All Android applications are composed of two things:
 - **Functionality** (code instructions)
 - **Data** (resources)
 - Resources include *text, strings, images and icons, audio files, videos, and other data* used by the application.
- ❑ Android resource files are **stored separately** from the java class files in the Android project
 - Most common resource types are **stored in XML**
 - You can also **store raw data** files and **graphics** as resources



Resource Directory Hierarchy

- ❑ All resources must be stored under the **/res** project directory in specially named subdirectories that must be lowercase:
- ❑ Default Android Resource Directories:

Resource Subdirectory	Purpose
/res/drawable-*/	Graphics Resources
/res/layout/	User Interface Resources
/res/values/	Strings, Color Values, etc



Accessing Resources

- ❑ Android Studio detects new resources when you add them to the appropriate project resource directory under **/res** automatically.
- ❑ Resources are compiled, resulting in the generation of the **R.java** file, which enables you to access your resources programmatically.
 - located in:
app\build\generated\source\r\debug\com\example\inika\simplelifecycletest
- ❑ All **resource IDs** are defined in your project's **R** class
- ❑ A resource ID is always composed of:
 - The *resource type*
 - The *resource name*
- ❑ Example: R.string.hello
 - **string** is the resource **type**
 - **hello** is the resource **name**.



Accessing Resources

- ❑ You can use a resource in code by passing the resource ID as a method parameter.
- ❑ For example, you can set an `ImageView` to use the `res/drawable/myimage.png` resource using **`setImageResource()`** method:

```
ImageView imageView = (ImageView)  
    findViewById(R.id.myimageview);  
imageView.setImageResource(R.drawable.myimage);
```



Accessing Resources from XML

- ❑ You can define values for some XML attributes and elements using a reference to an existing resource.
 - Necessary when creating layout files, to supply strings and images for your widgets.
- ❑ For example, if you add a Button to your layout, you should **use a string resource for the button text**:

<Button

```
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/submit" />
```



Managing Activity Transitions with Intents

in Android, you navigate b/t activities through what is known as an intent

- ❑ Android applications can have **multiple entry points**.
- ❑ There is **no main() function**, such as you find in iPhone development.
- ❑ A specific Activity can be designated as the main Activity to launch by default within the **AndroidManifest.xml** file
- ❑ Launching a New **Activity by Class Name**
 - You can start activities in several ways:
 - The simplest method is to use the Application **Context** object to call the **startActivity()** method, which takes a single parameter, an **Intent** object

```
startActivityForResult(  
    new Intent(this,  
    VRActivity.class),  
    requestCode);
```

In add, to passing in an Intent obj, need req code because multiple activities may be started at same time, need way to identify which activity returned result.

NOTE: if using req code == -1, no result will be returned



Managing Activity Transitions with Intents

- ❑ **Intents** allow sending or receiving data from and to other activities or services.
- ❑ Intents are objects of type "**android.content.Intent**" and are used to send asynchronous messages within your application or between applications.
- ❑ The following code uses Intent to **launch an Activity** named ActivityB by its class:

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

- ❑ You can use the **Intent** structure to **pass data between Activities**



Creating Intents with Action and Data

- ❑ Intent objects are composed of two main parts:
 - the ***action*** to be performed
 - the ***data*** to be acted upon
- ❑ You can also specify action/data pairs using **Intent Action** types and **Uri** objects
- ❑ The most common action types are defined in the Intent class, including ACTION_MAIN (describes the main entry point of an Activity) and ACTION_EDIT (used in conjunction with a Uri to the data edited)

```
Intent data = new Intent();
//---get the EditText view---
EditText txt_username =
    (EditText) findViewById(R.id.txt_username);
//---set the data to pass back---
data.setData(Uri.parse(
    txt_username.getText().toString()));
setResult(RESULT_OK, data);
//---closes the activity---
finish();
```

The setResult() method sets a result code (either RESULT_OK or RESULT_CANCELLED) and the data (an Intent object) to be returned back to the calling activity. The finish() method closes the activity and returns control to the calling activity.

In the calling activity, you need to implement the onActivityResult() method, which is called whenever an activity returns:

```
public void onActivityResult(int requestCode, int resultCode,
    Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this, data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```



Launching an Activity Belonging to Another Application

- ❑ Here is an example of how to create a simple Intent with a predefined Action (**ACTION_DIAL**) to launch the Phone Dialer with a specific phone number to dial in the form of a simple Uri object:

```
Uri number = Uri.parse(tel:5555551212);  
Intent dial = new Intent(Intent.ACTION_DIAL,  
    number);  
startActivity(dial);
```




Passing Additional Information Using Intents

- ❑ You can also include additional data in an Intent.
 - The **Extras** property of an Intent is stored in a **Bundle** object.
- ❑ The Intent class also has a number of helper methods for **getting and setting name/value pairs** for many common data types.
- ❑ For example, the following Intent includes two extra pieces of information - a string value and a boolean:

```
Intent intent = new Intent(this, MyActivity.class);  
intent.putExtra("SomeStringData", "Foo");  
intent.putExtra("SomeBooleanData", false);
```



Intents Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title" />
    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="callIntent"
        android:text="@string/submit1" >

</Button>

    <Button
        android:id="@+id/Button02"
        android:layout_width="114dp"
        android:layout_height="wrap_content"
        android:onClick="callIntent"
        android:text="@string/submit2"
        android:width="100dp" >

</Button>
</LinearLayout>
```



Intents Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="test.samples"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".AndroidTestActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.CALL_PRIVILEGED"></uses-permission>
    <uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.CAMERA"></uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>

</manifest>
```



Intents Example

```
package test.samples;
```

```
import android.app.Activity;
```

```
import android.os.Bundle;
```

```
import android.content.Intent;
```

```
import android.net.Uri;
```

```
import android.view.View;
```

```
public class AndroidTestActivity extends Activity {
```

```
    /** Called when the activity is first created. */
```

```
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
```

```
    }
```



Intents Example

```
public void callIntent(View view) {  
    Intent intent = null;  
    switch (view.getId()) {  
        case R.id.Button01:  
            intent = new Intent(Intent.ACTION_VIEW,  
                Uri.parse("http://centennialcollege.ca"));  
            startActivity(intent);  
            break;  
        case R.id.Button02:  
            intent = new Intent(Intent.ACTION_CALL,  
                Uri.parse("tel:(416)289-5000"));  
            startActivity(intent);  
            break;  
        default:  
            break;  
    }  
}  
} // end of AndroidTestActivity
```

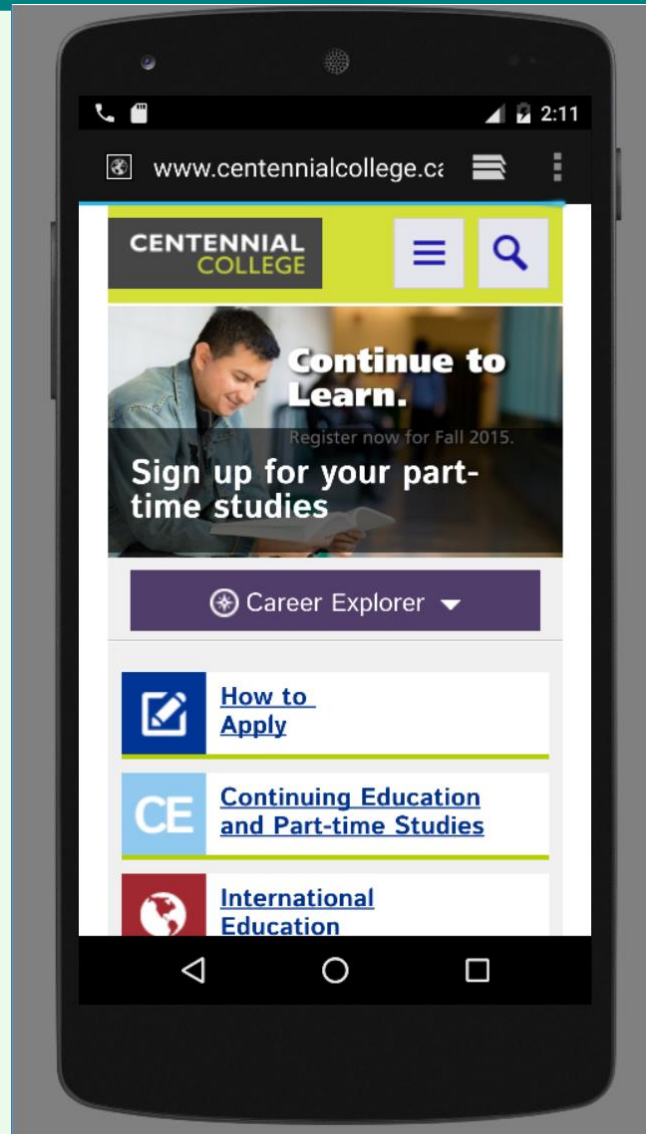


Intents Example



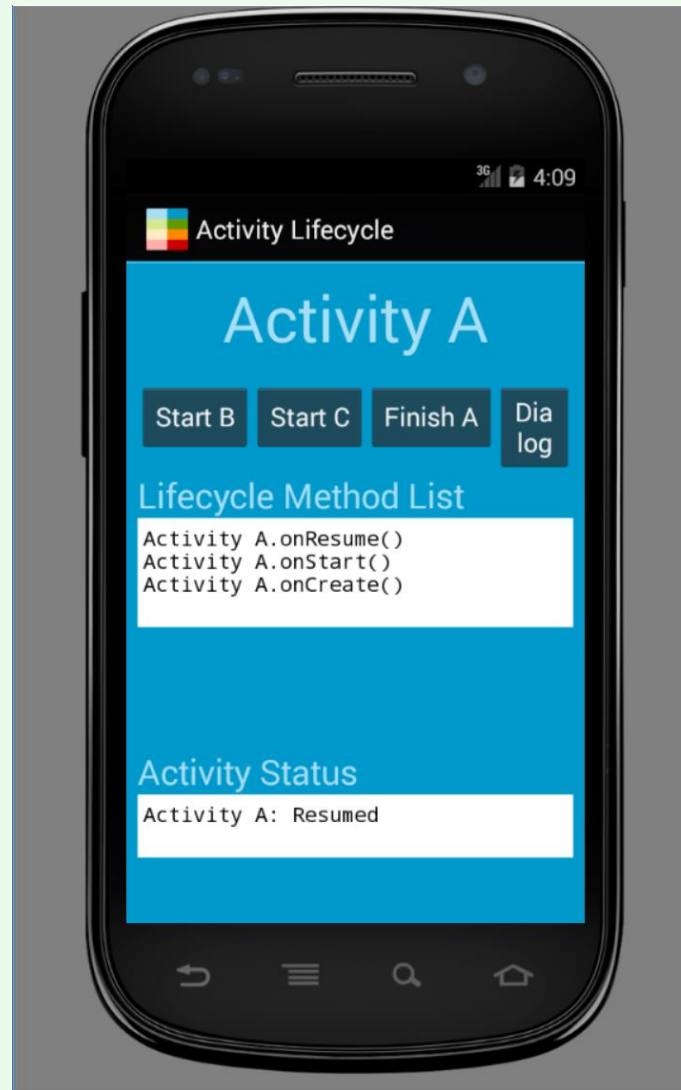


Intents Example





Activity Life Cycle example with Intents





References

- ❑ Textbook
- ❑ Reference book
- ❑ Android Documentation:
<https://developer.android.com/guide/components/activities.html>