Activities

Learning Objectives

After studying this module, the students will be able to

- Understand the activity and life cycle of an activity
- Configure activity in AndroidManifest.xml file
- Know activity transition

Introduction

Generally, a program is defined in terms of functionality and data, and an Android application is not an exception. It performs processing, show information on the screen, and takes data from a variety of sources.

To Develop Android applications for mobile devices with resource constraint requires a systematic understanding of the application lifecycle. Important terminology for application building blocks terms are Context, Activity, and Intent. This module introduces you with the most important components of Android applications and provides you with a more detailed understanding of how Android applications function and interact with one another.

The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main () method, the Android system initiates code in an activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

This document introduces the concept of activities, and then provides some lightweight guidance about how to work with them.

What is activity?

The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user journey often begins non-deterministically. For instance, if you open an email app from your home screen, you might see a list of emails. By contrast, if you are using a social media app that then launches your email app, you might go directly to the email app's screen for composing an email.

The Activity class is designed to facilitate this paradigm. When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole. In this way, the activity serves as the entry point for an app's interaction with the user. You implement an activity as a subclass of the Activity class.

An activity provides the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. Generally, one activity implements one screen in an app. For instance, one of an app's activities may implement a Preferences screen, while another activity implements a Select Photo screen.

Most apps contain multiple screens, which means they comprise multiple activities. Typically, one activity in an app is specified as the main activity, which is the first screen to appear when the user launches the app. Each activity can then start another activity in order to perform different actions. For example, the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

Although activities work together to form a cohesive user experience in an app, each activity is only loosely bound to the other activities; there are usually minimal dependencies among the activities in an app. In fact, activities often start up activities belonging to other apps. For example, a browser app might launch the Share activity of a social-media app.

To use activities in your app, you must register information about them in the app's manifest, and you must manage activity lifecycles appropriately. The rest of this document introduces these subjects.

Configuring the AndroidManifest.xml

For your app to be able to use activities, you must declare the activities, and certain of their attributes, in the manifest.

Declare activities: To declare your activity, open your manifest file and add an <activity> element as a child of the <application> element. For example:

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

The only required attribute for this element is android:name, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme.

Declare intent filters: Intent filters are a very powerful feature of the Android platform. They provide the ability to launch an activity based not only on an explicit request, but also an implicit one. For example, an explicit request might tell the system to "Start the Send Email activity in the Gmail app". By contrast, an implicit request tells the system to

“Start a Send Email screen in any activity that can do the job." When the system UI asks a user which app to use in performing a task, that’s an intent filter at work.

You can take advantage of this feature by declaring an <intent-filter> attribute in the <activity> element. The definition of this element includes an <action> element and, optionally, a <category> element and/or a <data> element. These elements combine to specify the type of intent to which your activity can respond. For example, the following code snippet shows how to configure an activity that sends text data, and receives requests from other activities to do so:

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

In this example, the <action> element specifies that this activity sends data. Declaring the <category> element as DEFAULT enables the activity to receive launch requests. The <data> element specifies the type of data that this activity can send. The following code snippet shows how to call the activity described above

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
// Start the activity
startActivity(sendIntent);
```

If you intend for your app to be self-contained and not allow other apps to activate its activities, you don't need any other intent filters. Activities that you don't want to make available to other applications should have no intent filters, and you can start them yourself using explicit intents.

Declare permissions: You can use the manifest's <activity> tag to control which apps can start a particular activity. A parent activity cannot launch a child activity unless both activities have the same permissions in their manifest. If you declare a <uses-permission> element for a particular activity, the calling activity must have a matching<uses-permission> element.

For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

```
<manifest>
<activity android:name="...."
                    android:permission=”com.google.socialapp.permission.SHARE_POST”
/>
```

Then, to be allowed to call SocialApp, your app must match the permission set in SocialApp's manifest:

```
<manifest>
<uses-permission  android:name="com.google.socialapp.permission.SHARE_POST"  />
</manifest>
```

Check your progress-1

a) When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole (True/False)

b) Most apps contain multiple screens, which means they comprise multiple _____

(A) Activities (B) Services (C) Contexts (D) Intents

c) You can use activities in your app, without registering the information about them in the app's manifest,

d) By default, the activity created for you contains the ________ event

e) The only required attribute for <activity> element is __________

Life Cycle of an Activity

The Activity class is an important for application's whole lifecycle. Android applications can be multi-process, and the multiple applications to run concurrently if memory and processing power is available. Applications can have background processes, and applications can be interrupted/paused when events such as message or phone calls occur. There can be only one active application visible to the user at a time or in other words only a single Activity is in the foreground at any given time.

Activities in the Android operating system are managed using an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running/foreground activity the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

Activity States

An activity has essentially four states:

State	Description
Active or running	When an activity is in the foreground of the screen (at the top of the stack).
Paused	If an activity has lost focus but is still visible, it is paused. A paused activity maintains all state and member information and remains attached to the window manager, but can be killed by the system in extreme low memory situations.
Stopped	If an activity is completely hidden by another activity, it is stopped. It still retains all state and member information, it will often be killed by the system when memory is needed elsewhere.

Destroyed	If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

Table-1

Activity Events

The Activity base class defines a series of events that governs the life cycle of an activity. The Activity class defines the following events:

Event	Description
onCreate()	Called when the activity is first created
onStart()	Called when the activity becomes visible to the user
onResume()	Called when the activity starts interacting with the user
onPause()	Called when the current activity is being paused and the previous activity is being resumed
onStop()	Called when the activity is no longer visible to the user
onDestroy()	Called before the activity is destroyed by the system
onRestart()	Called when the activity has been stopped and is restarting again

Table-2

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen.

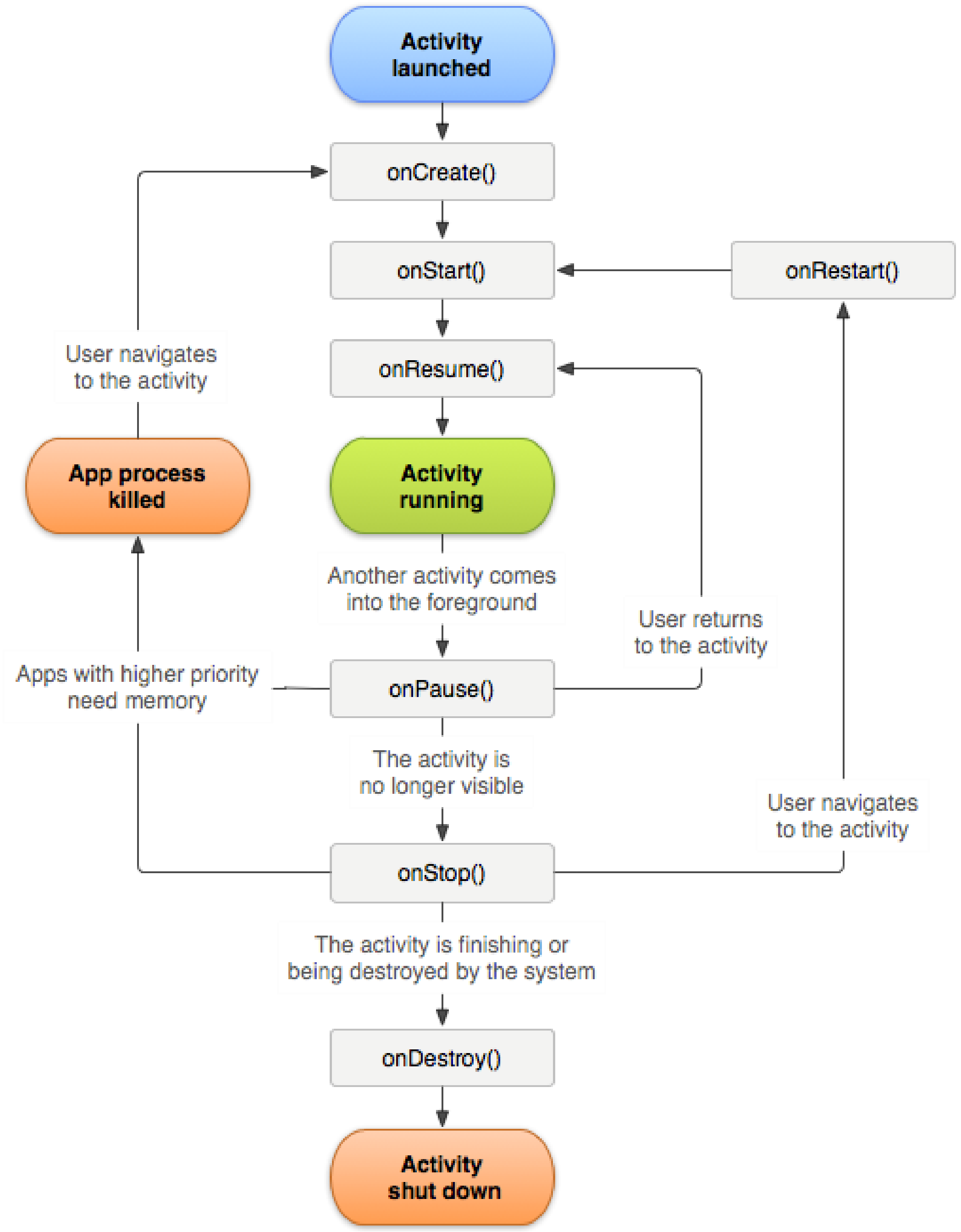

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure-1: important state paths of an Activity

The above figure shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.

Understanding Life Cycle of an Activity

The best way to understand the various stages experienced by an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

1. Create a New Android Studio Project with project name ActivityDemo and Main Activity name as MainActivity
2. In the MainActivity.java file, add the following statements in bold:

```
package in.edu.baou.activitydemo;

import android.util.Log;

public class MainActivity extends ActionBarActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d("Event", "In the onCreate() event");
   }
   public void onStart()
   {
      super.onStart();
      Log.d("Event", "In the onStart() event");
   }
   public void onRestart()
   {
      super.onRestart();
      Log.d("Event", "In the onRestart() event");
   }

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

3. Press Shift+F10 or ‘Run App’ button in taskbar. It will launch following dialog box. Press OK.

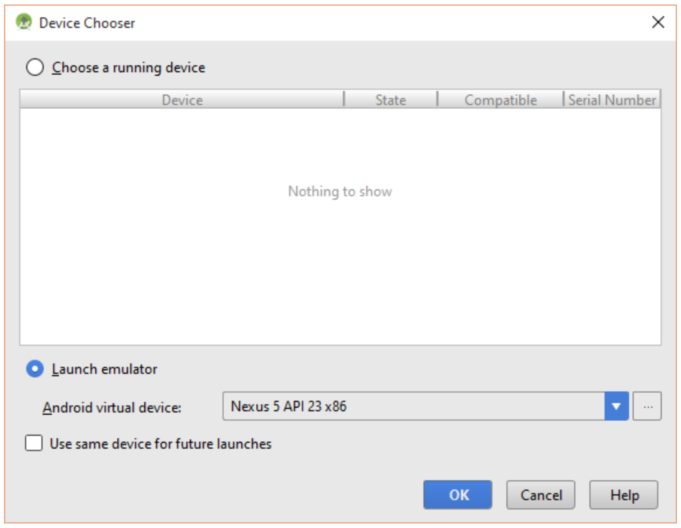
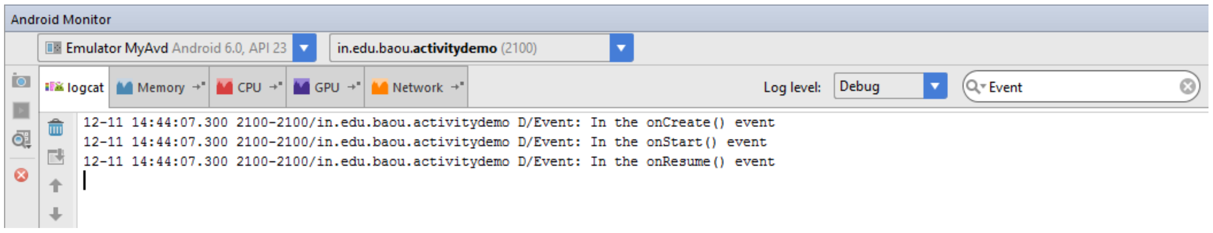


Figure-2

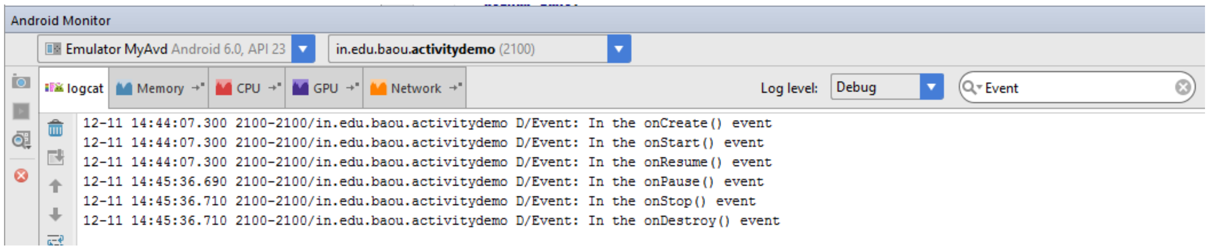
4. When the activity is first loaded, you should see the following in the LogCat window.


Android Monitor
Emulator MyAvd Android 6.0, API 23
in.edu.baou.activitydemo (2100)
logcat
Memory
CPU
GPU
Network
Log level: Debug
Event

```
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
```

Figure-3

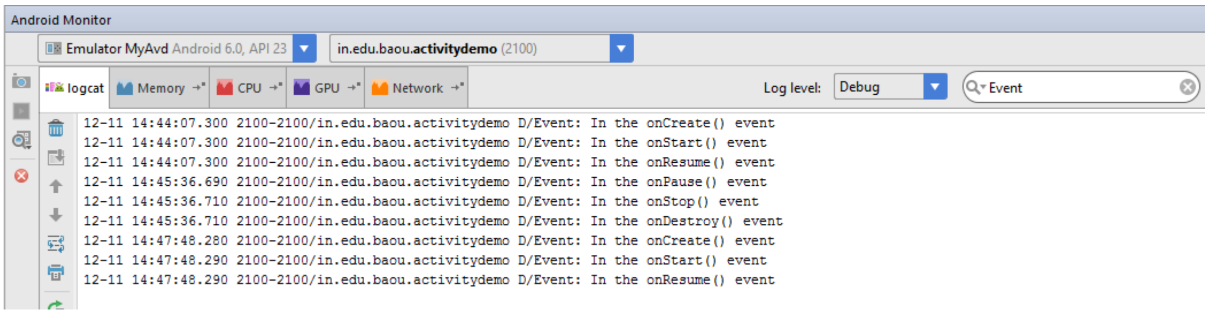
5. Now press the back button on the Android Emulator, observe that the following is printed:


Android Monitor
Emulator MyAvd Android 6.0, API 23
in.edu.baou.activitydemo (2100)
logcat
Memory
CPU
GPU
Network
Log level:
Debug
Event

```
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 14:45:36.690 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 14:45:36.710 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
12-11 14:45:36.710 2100-2100/in.edu.baou.activitydemo D/Event: In the onDestroy() event
```

Figure-4

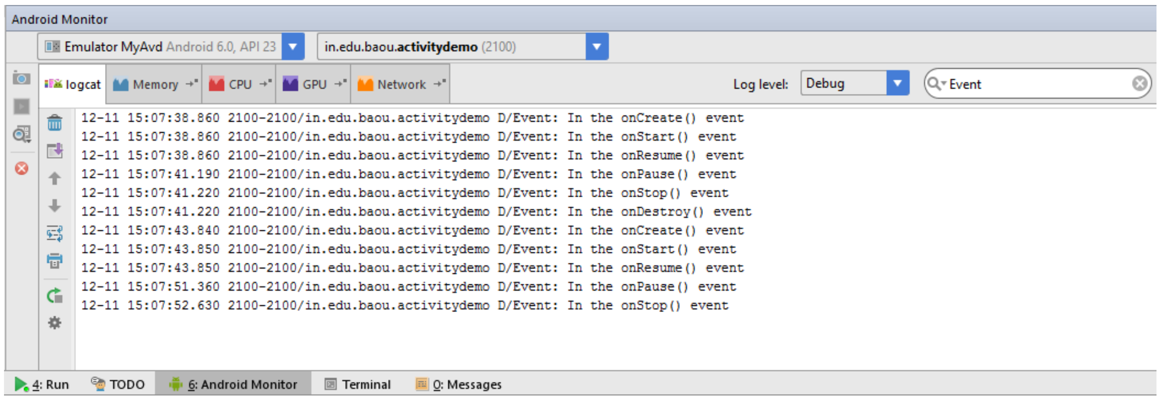
6. Click the Home button and hold it there. Click the ActivityDemo icon and observe the following:


Android Monitor
Emulator MyAvd Android 6.0, API 23
in.edu.baou.activitydemo (2100)
logcat
Memory
CPU
GPU
Network
Log level:
Debug
Event

```
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 14:44:07.300 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 14:45:36.690 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 14:45:36.710 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
12-11 14:45:36.710 2100-2100/in.edu.baou.activitydemo D/Event: In the onDestroy() event
12-11 14:47:48.280 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 14:47:48.290 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 14:47:48.290 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
```

Figure-5

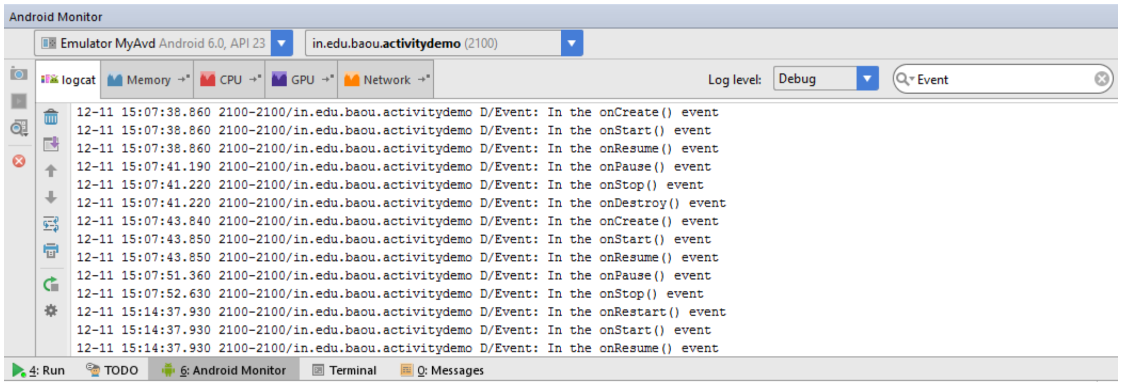
7. On Android Emulator from notification area open settings on so that the activity is pushed to the background. Observe the output in the LogCat window:



```
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 15:07:41.190 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 15:07:41.220 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
12-11 15:07:41.220 2100-2100/in.edu.baou.activitydemo D/Event: In the onDestroy() event
12-11 15:07:43.840 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 15:07:43.850 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 15:07:43.850 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 15:07:51.360 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 15:07:52.630 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
```

Figure-6

8. Notice that the onDestroy() event is not called, indicating that the activity is still in memory. Exit the settings by pressing the Back button. The activity is now visible again. Observe the output in the LogCat window:



```
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 15:07:38.860 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 15:07:41.190 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 15:07:41.220 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
12-11 15:07:41.220 2100-2100/in.edu.baou.activitydemo D/Event: In the onDestroy() event
12-11 15:07:43.840 2100-2100/in.edu.baou.activitydemo D/Event: In the onCreate() event
12-11 15:07:43.850 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 15:07:43.850 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
12-11 15:07:51.360 2100-2100/in.edu.baou.activitydemo D/Event: In the onPause() event
12-11 15:07:52.630 2100-2100/in.edu.baou.activitydemo D/Event: In the onStop() event
12-11 15:14:37.930 2100-2100/in.edu.baou.activitydemo D/Event: In the onRestart() event
12-11 15:14:37.930 2100-2100/in.edu.baou.activitydemo D/Event: In the onStart() event
12-11 15:14:37.930 2100-2100/in.edu.baou.activitydemo D/Event: In the onResume() event
```

Figure-7

Please note that the onRestart() event is now fired, followed by the onStart() and onResume() events.

This application uses logging feature of Android. To add logging support to ActivityDemo app, edit the file MainActivity.java to add the following import statement for the Log class:

import android.util.Log;

Logging is a valuable resource for debugging and learning Android. Android logging features are in the Log class of the android.util package. Some helpful methods in the android.util.Log class are shown in Table. We have used Log.d() method to print message in LogCat Window when particular event of activity fired.

Method	Purpose
Log.e()	Log errors
Log.w()	Log warnings
Log.i()	Log information messages
Log.d()	Log debug messages
Log.v()	Log verbose messages

Table-3

Context

As the name suggests, it is the context of current state of the application/object. It lets newly created objects understand what has been going on. Typically you call it to get information regarding other part of your program (activity, package/application). The application Context is the central location for all top-level application functionality. The Context class can be used to manage application-specific configuration details as well as application-wide operations and data. Use the application Context to access settings and resources shared across multiple Activity instances.

Retrieving the Application Context

You can get the context by invoking **getApplicationContext(), getContext(), getBaseContext()** or this (when in the activity class). You can retrieve the Context for the current process using the getApplicationContext() method, like this: **Context context = getApplicationContext();**

Uses of the Application Context

After you have retrieved a valid application Context, it can be used to access application-wide features and services. Typical uses of context are:

1) **Creating new views, adapters, listeners object**

```
TextView tv = new TextView(getContext());
ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), ...);
```

2) **Retrieving Application Resources:** You can retrieve application resources using the getResources() method of the application Context. The most straightforward way to retrieve a resource is by using its resource identifier, a unique number automatically generated within the R.java class.The following example retrieves a String instance from the application resources by its resource ID:

```
String greeting = getResources().getString(R.string.settings);
```

3) **Retrieving Shared Application Preferences:** You can retrieve shared application preferences using the getSharedPreferences() method of the application Context. The

SharedPreferences class can be used to save simple application data, such as configuration settings.

4) **Accessing Other Application Functionality Using Context:** The application Context provides access to a number of other top-level application features.
Here are a few more things you can do with the application Context:
   - Launch Activity instances
   - Inspect and enforce application permissions
   - Retrieve assets packaged with the application
   - Request a system service (for example, location service)
   - Manage private application files, directories, and databases

Activity Transition

In the course of the lifetime of an Android application, the user might transition between a numbers of different Activity instances. At times, there might be multiple Activity instances on the activity stack. Developers need to pay attention to the lifecycle of each Activity during these transitions.

Some Activity instances such as the application splash/startup screen are shown and then permanently discarded when the Main menu screen Activity takes over. The user cannot return to the splash screen Activity without re-launching the application.

Other Activity transitions are temporary, such as a child Activity displaying a dialog box, and then returning to the original Activity (which was paused on the activity stack and now resumes). In this case, the parent Activity launches the child Activity and expects a result.

Transitioning between Activities with Intents: As previously mentioned, Android applications can have multiple entry points. There is no main() function, such as you find in iPhone development. Instead, a specific Activity can be designated as the main Activity to launch by default within the AndroidManifest.xml file; Other Activities might be designated to launch under specific circumstances. For example, a music application might designate a generic Activity to launch by default from the Application menu, but also define specific alternative entry point Activities for accessing specific music playlists by playlist ID or artists by name.

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.

Intent (android.content.Intent) is an asynchronous message mechanism used by the Android operating system to match task requests with the appropriate Activity or Service (launching it, if necessary) and to dispatch broadcast Intents events to the system at large. For now, though, we focus on Intents and how they are used with Activities. The following line of code calls the startActivity() method with an explicit Intent. This Intent requests the launch of the target Activity named MyDrawActivity by its class. This class is implemented elsewhere within the package.

```
startActivity(new Intent(getApplicationContext(),
MyDrawActivity.class));
```

This line of code might be sufficient for some applications, which simply transition from one Activity to the next. However, you can use the Intent mechanism in a much more robust manner. For example, you can use the Intent structure to pass data between Activities.

Creating Intents with Action and Data: You've seen the simplest case to use Intent to launch a class by name. Intents need not specify the component or class they want to launch explicitly. Instead, you can create an Intent Filter and register it within the Android Manifest file. The Android operating system attempts to resolve the Intent requirements and launch the appropriate Activity based on the filter criteria.

The guts of the Intent object are composed of two main parts: the action to be performed and the data to be acted upon. You can also specify action/data pairs using Intent Action types and Uri objects. An Uri object represents a string that gives the location and name of an object. Therefore, an Intent is basically saying “do this” (the action) to “that” (the Uri describing what resource to do the action to). 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).You also find Action types that generate integration points with Activities in other applications, such as the Browser or Phone Dialer.

Launching an Activity Belonging to another Application: Initially, your application might be starting only Activities defined within its own package

However, with the appropriate permissions, applications might also launch external Activities within other applications. For example, a Customer Relationship Management (CRM) application might launch the Contacts application to browse the Contact database, choose a specific contact, and return that Contact’s unique identifier to the CRM application for use.

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);
```

Let us sum up

In this module you have learn all about android activity, important terminology. You have understood the life cycle of an activity through practical example and you have also lean about how to configure activity in AndroidManifest.xml file at the end of module we have discuss activity transition.

Further Reading

- http://developer.android.com/guide/appendix/g-app-intents.html.

Activity

- Write android activity using logging feature to demonstrate activity life cycle

Acknowledgement: “The content in this module is modifications based on work created and shared by the Android Open-Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.”