

Introduction of Key Concepts of Android

CONTENTS

3.1	App Components
3.1.1	Activities
3.1.2	Services
3.1.3	Content Providers
3.1.4	Intents
3.2	App Resources
3.3	App Manifest
3.3.1	Elements
3.3.2	Attributes
3.3.3	Declaring Class Names
3.3.4	Multiple Values
3.3.5	Resource Values
3.3.6	String Values

Understand key concepts of Android is a basic requirement for designing Android mobile apps. In this chapter, we introduce some basic concepts of Android, including the app components, app resources, and app manifest. Students will be able to answer the following questions after reading this chapter.

1. What is an activity in Android?
2. Can we directly save resource files inside the res/directory?
3. What is an *APP MANIFEST*?

3.1 APP COMPONENTS

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user, and some depend on each other, but each one exists as

its own entity and plays a specific role. Each one is a unique building block that helps define your app's overall behavior.

The following subsections represent four types of app components, which include activities, services, content providers, and intents.

3.1.1 Activities

An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture. You can find more information about activities at <https://developer.android.com/guide/components/activities.html>.

3.1.2 Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. You can find more information about content providers at <https://developer.android.com/reference/android/app/Service.html>.

3.1.3 Content Providers

A content provider manages a shared set of app data. You can store the data in the file system, a SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as `ContactsContract.Data`) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the Note Pad sample app uses a content provider to save notes. You can find more information about content provider at: <https://developer.android.com/reference/android/content/ContentProvider.html>

3.1.4 Intents

An intent is a mechanism for describing a specific action, such as “pick a photo,” “phone home,” or “open the pod bay doors.” In Android, just about everything goes through intents, so you have plenty of opportunities to replace or reuse components. For example, there is an intent for “send an email.” If your application needs to send mail, you can invoke that intent. Or, if you are writing a new email application, you can register an activity to handle that intent and replace the standard mail program. The next time somebody tries to send an email, they'll get the option to use your program instead of the standard one. You can find more information about intents at <https://developer.android.com/guide/components/intents-filters.html>.

A unique aspect of the Android system design is that any app can start another app's component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it, instead of developing

an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no `main()` function, for example).

Since the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. The Android system, however, can. So, to activate a component in another app, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

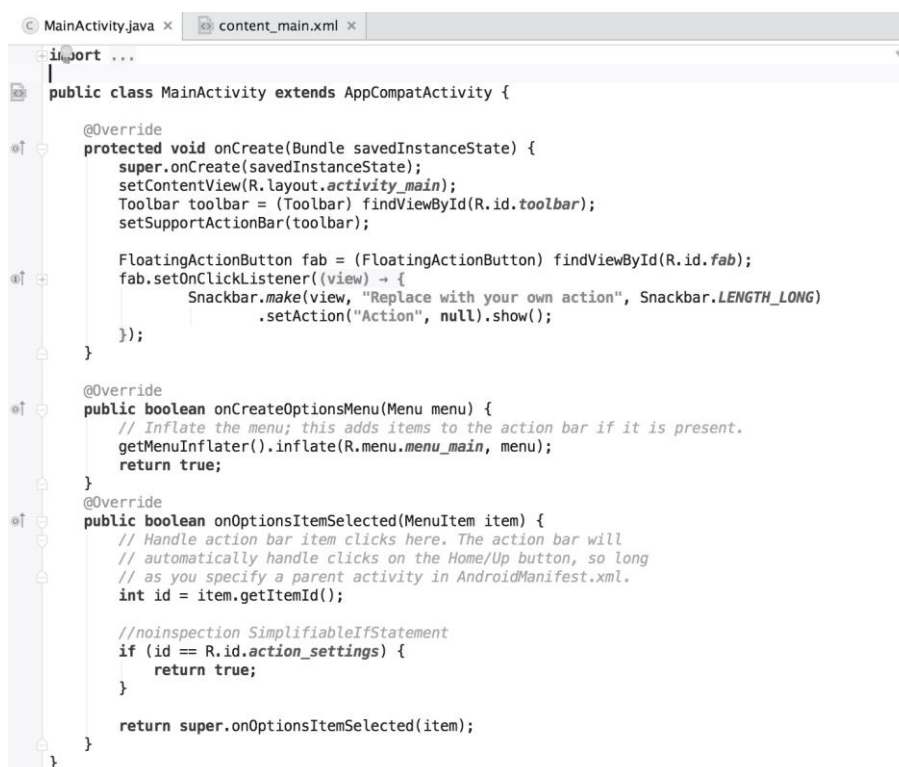
Intents can be used to activate activities and services, but content providers are activated when targeted by a request from a *ContentResolver*. There are separate methods for activating each type of component:

1. You can start an activity (or give it something new to do) by passing an *Intent* to `startActivity()` or `startActivityForResult()` (when you want the activity to return a result).
2. You can start a service (or give new instructions to an ongoing service) by passing an *Intent* to `startService()`. Or you can bind to the service by passing an *Intent* to `bindService()`.

3. You can perform a query to a content provider by calling *query()* on a *ContentResolver*.

Fig. 3.1 is the default *AndroidManifest.xml* generated by the Integrated Development Environment (IDE) after we create a blank Android application. The *Main Activity* is the only activity in this project, and it is a subclass of *Activity*, as “class *Main Activity* extends *Activity*” shown. In *Main Activity*, we need to implement callback methods that the system calls when the activity transitions between various states of its life cycle.

The *onCreate()* method is indispensable, and it will be called when the *MainActivity* is created. Within the implementation of the *onCreate()* method, we should initialize the essential components of this activity. We must call *setContentView()* to define the layout for this activity’s user interface. Although these processes are implemented by IDE, it is necessary for us to have this knowledge.



```
import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

Figure 3.1 Default *MainActivity.java*.