ANDROID FUNDAMENTAL COMPONENT

ROMI FADILLAH RAHMAT

ACTIVITY

- An activity is a user interface concept. An activity usually represents a single screen in your application.
- It generally contains one or more views, but it doesn't have to.
- Activities provide the reusable, interchangeable parts of the flow of UI components across Android applications.
- A well-designed activity is responsible for managing a single UI page and each has its own unique name.
- Users move from page to page in a web application by following the links, while in Android applications users may find their interaction invoked by an intent.

ACTIVITY

- Just as in the world of web applications, some servlets provide UIs and others provide APIs for services, so, in the Android world, activities provide UIs and the Service and ContentProvider classes, introduced in a moment, provide programmatic access to services.
- Understanding this architectural similarity will help you design Android applications that effectively use the Android Framework.

ACTIVITY

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

ACTIVITY

- When the system starts this activity it calls the constructor for TestActivity, a subclass of Activity, and then calls its onCreate method. This causes the view hierarchy described in the *main.xml* file to load and display.
- The onCreate method kicks off the life cycle of the Activity
- We use the word *activity* to refer to instances of the Activity class, much in the way that *object* is used to refer to instances of classes

INTENT

- “An intent is an abstract description of an operation to be performed.” (developer.android.com)
- A *bit like* a method call
- Two flavours: explicit and implicit
  - An **explicit** Intent specifies exactly which Activity should be stated
  - An **implicit** Intent is more declarative: it what the Activity should do
  - The system will then search for Activities that match by checking the Intent filters
  - Example: opening a Web Page (more on this later)

INTENT

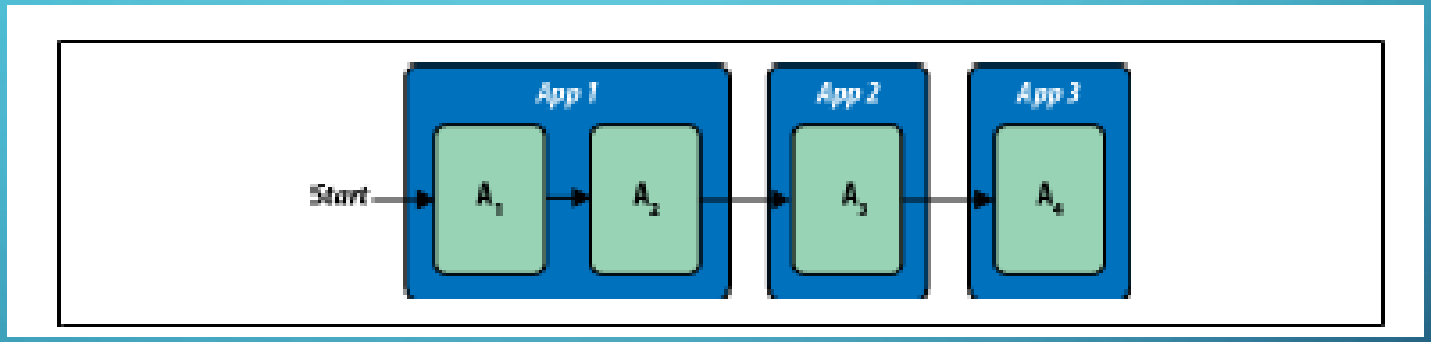
- An intent generically defines an “intention” to do some work. Intents encapsulate several concepts, so the best approach to understanding them is to see examples of their use.
- You can use intents to perform the following tasks:
  - Broadcast a message
  - Start a service
  - Launch an activity
  - Display a web page or a list of contacts
  - Dial a phone number or answer a phone call

INTENT

- How does one activity invoke another, and pass information about what the user wants to do? The unit of communication is the Intent class.
- An intent represents an abstract description of an operation that one activity requires another activity to perform, such as taking a picture.
- Intents form the basis of a system of loose coupling that allows activities to launch one another, and provide results. When an application dispatches an intent, it's possible that several different activities might be registered to provide the desired operation.

TASK

- Task = a chain of activities that often span more than one application, and, often, more than one process.



- It shows a task spanning three applications and multiple activities
- The chain of activities comprising this task spans three separate processes and heaps, and can exist independent of other tasks that may have started other instances of the same Activity subclasses.

ACTIVITY, TASK EXAMPLE

- Example of single task, made up of activities across applications

App	Activity	User's next action
Messaging	View list of messages	User taps on a message in the list
Messaging	View a message	User taps Menu→View Contact
Contacts	View a contact	User taps Call Mobile
Phone	Call the contact's mobile number	

ACTIVITY STATE TRANSITIONS AND METHODS

http://developer.android.com/training/basics/activity-lifecycle/starting.html

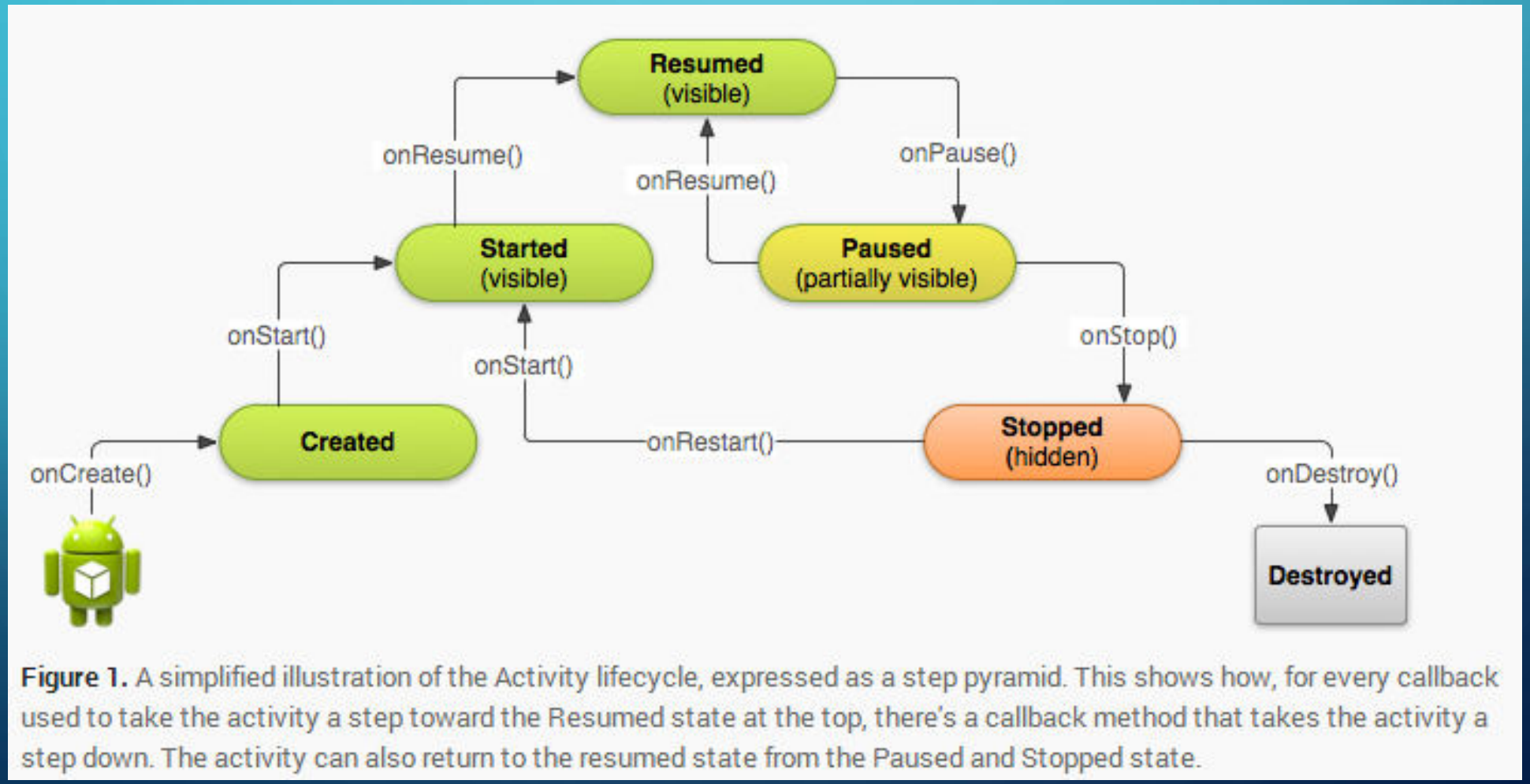
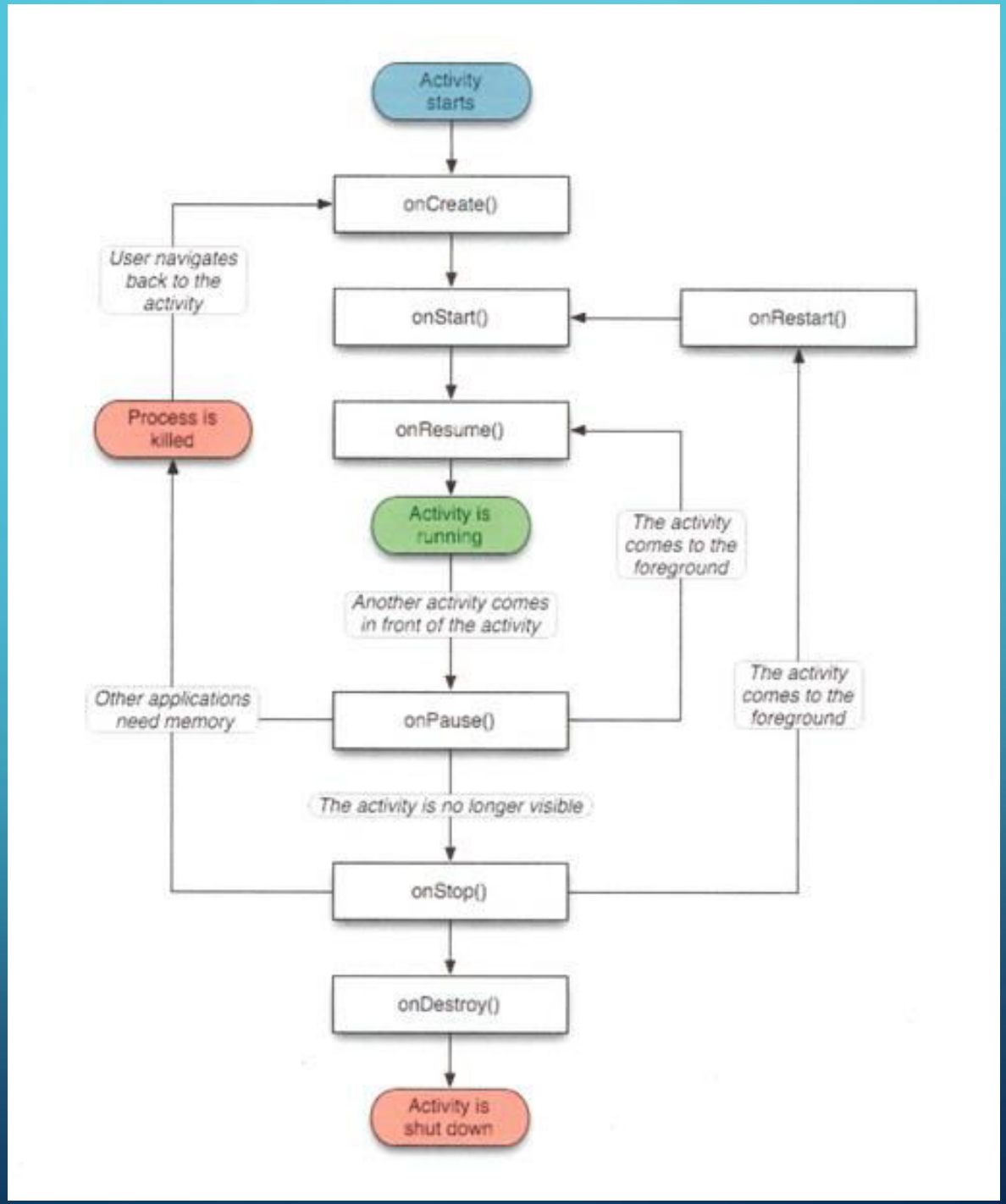


Figure 1. A simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

TIPS FOR STATE MANAGEMENT

- Save any important information frequently or immediately
  - Mobile device: the battery could die any time!
- Override onPause or onStop to save useful non-critical state (onDestroy is a bit too last-minute for most things)
- When overriding onCreate, check to see whether savedInstanceState is non-null

DESTROY()

HTTP://DEVELOPER.ANDROID.COM/TRAINING/BASICS/ACTIVITY-LIFECYCLE/STARTING.HTML

- "While the activity's first lifecycle callback is onCreate(), its very last callback is onDestroy(). The system calls this method on your activity as the final signal that your activity instance is being completely removed from the system memory.
- Most apps don't need to implement this method because local class references are destroyed with the activity and your activity should perform most cleanup during onPause() and onStop(). However, if your activity includes background threads that you created during onCreate() or other long-running resources that could potentially leak memory if not properly closed, you should kill them during onDestroy()."

SCREEN ORIENTATION

- Each time the screen is rotated, the current activity is destroyed, and then re-created
- Predefined onCreate() method **retrieves state of any View components (i.e. components that sub-class View; this eases the job of the programmer**
- This follows the call graph in fig 1.
- Rationale:
  - Typically a new layout may be needed, involving new resource allocation
  - Cleanest solution: always destroy and re-create
- Note: apps can specify to always operate in a particular orientation

MANAGING STATE BETWEEN ORIENTATION CHANGES

- Save state information to the Bundle by overriding onStop()
- Re-create state by extracting information from bundle in onCreate()
- Discussion question:
  - In my Lissajous App I didn't do this, yet state is preserved betweem orientation changes. Why?

SPECIFYING ORIENTATION ETC

- Following example specifies some other details in a fairly obvious way (easiest way to find this was via a Google search)

```
requestWindowFeature(Window.FEATURE_NO_TITLE);

setRequestedOrientation(
    ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

STARTING A NEW ACTIVITY

- Define a class that sub-classes Activity
- Add some GUI control to invoke it from the parent activity
- Listen for the relevant event, then launch a new Intent
- This will indirectly call the new Activity's method:
  - `onCreate(Bundle savedInstance)`
- The new activity will start and enter then Resumed state via the call graph shown previously

THE FOLLOWING EXAMPLE ADDS AN ACTIVITY TO PROVIDE INFORMATION ABOUT AN APP

- A menu item called “About” is added to the options menu
- We listen for **onOptionItemSelected** events within the main activity
- Create an **Intent**, then call **startActivity** with the Intent as an argument
- When the user has finished reading the HTML page, the back button can be used to return to the main app
  - This behaviour is automatic use of the “back stack”: no need to program it

ADDING THE MENU ITEM TO OUR MAIN ACTIVITY, LISTENING FOR IT AND LAUNCHING INTENT

```
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add("About");
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getTitle().equals("About")) {
        Intent intent =
            new Intent(this, AboutActivity.class);
        startActivity(intent);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
    …
```

SERVICE

- Services in Android resemble services you see in Windows or other platforms—they're background processes that can potentially run for a long time.
- Android defines two types of services: local services and remote services.
- Local services are components that are only accessible by the application that is hosting the service.
- Conversely, remote services are services that are meant to be accessed remotely by other applications running on the device.

- Once a service starts, it is likely to be available unless memory gets extremely constrained
- Example : A music-playing application would likely be implemented as a service to continue to play music while a user might be viewing web pages.
- Services also allow applications to share functions through long-term connections.

CONTENT PROVIDER

- Data sharing among mobile applications on a device is common.
- Therefore, Android defines a standard mechanism for applications to share data (such as a list of contacts) without exposing the underlying storage, structure, and implementation.
- Through content providers, you can expose your data and have your applications use data from other applications.

CONTENT PROVIDER

- To use a ContentProvider, you specify a URI and how to act on referenced data.
- Content provider operations, which provide the well-known quartet of basic data handling activities: create (insert), read (query), update, and delete.
- Insert - The insert method of the ContentProvider class is for the purpose of inserting a new records into the database.
- Query - The query method of the ContentProvider class is for the purpose of returning a set of records in a specialized collection class called Cursor.

- Update - The update method of the ContentProvider class is for the purpose of replacing records in the database with updated records.
- Delete - The delete method of the ContentProvider class is for the purpose of removing matching records from the database.

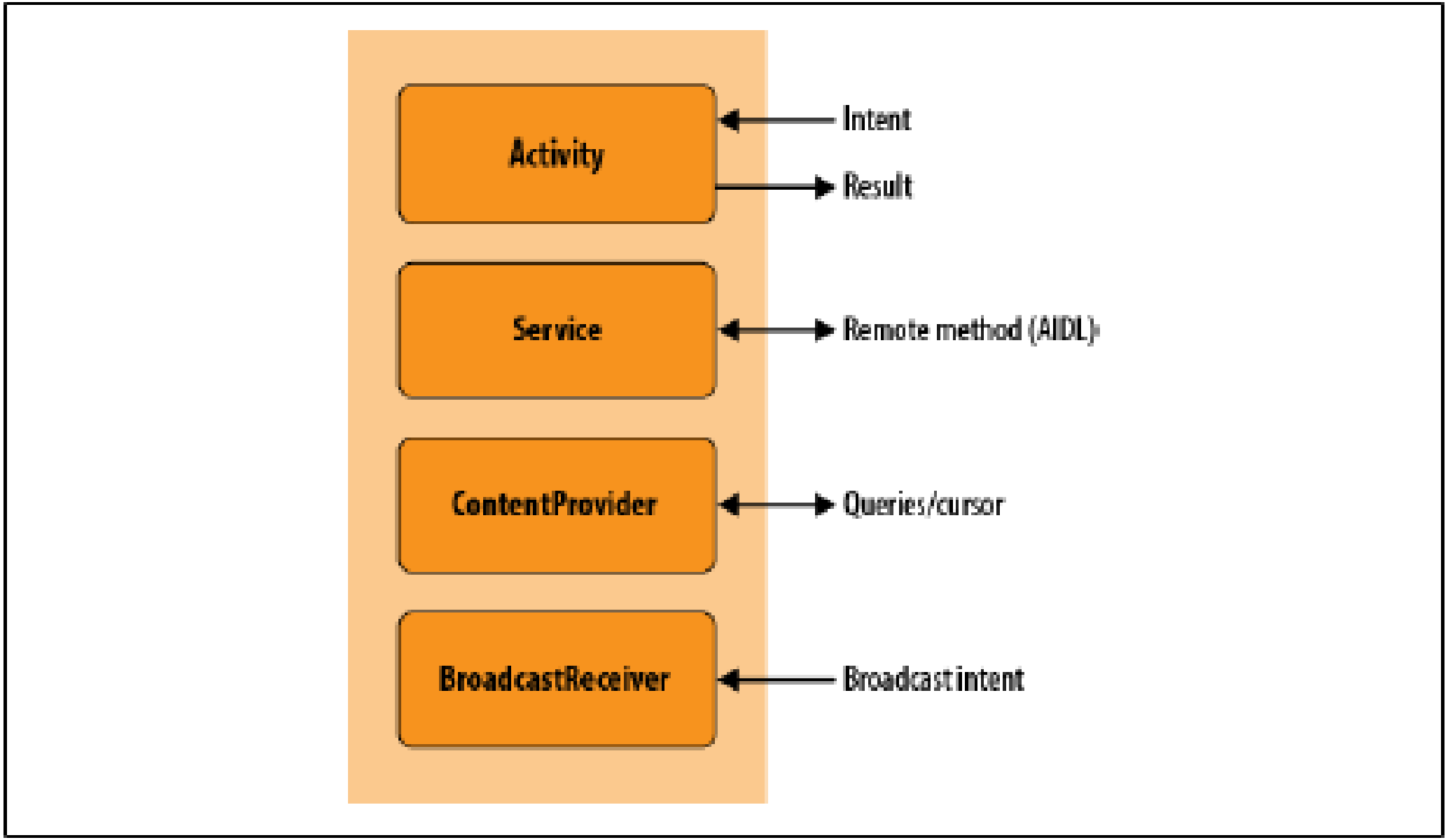
- content providers allow developers to efficiently share entire SQL databases across processes: instead of sharing just objects, content providers manage entire SQL tables.

BROADCAST RECEIVER

- The BroadcastReceiver class implements another variant of Android's high-level interprocess communication mechanism using Intent objects.
- It is similar to an activity but does not have its own user interface. All of the broadcast receivers registered for a given intent receive that intent whenever it is broadcast.
- A typical use for a broadcast receiver might be to receive an alarm that causes an app to become active at a particular time.

APPLICATION MANIFEST

- In addition to data that is used by a running application, an application also needs a way to describe the environment it expects: its name, the intents it registers, the permissions it needs, and other information that describes the app to the Android system. This data is stored in a file called the *manifest*.
- The four components of Android applications—Activity, Service, ContentProvider, and BroadcastReceiver—provide the foundation of Android application development
- To make use of any of them, an application must include corresponding declarations in its *AndroidManifest.xml* file


Activity
Intent
Result
Service
Remote method (AIDL)
ContentProvider
Queries/cursor
BroadcastReceiver
Broadcast intent

APPLICATION MANIFEST

- Android requires applications to explicitly describe these things in an XML file called AndroidManifest.xml.
- Here, applications declare the presence of content providers, services, required permissions, and other elements.
- Manifest data is also available to a running application through its context.
- The manifest file organizes an Android application into a well-defined structure that is shared by all applications and enables the Android operating system to load and execute them in a managed environment.

ANDROIDMANIFEST.XML

- AndroidManifest.xml, which is similar to the web.xml file in the J2EE world, defines the contents and behavior of your application.
- For example, it lists your application's activities and services, along with the permissions the application needs to run.

ANDROID RUNTIME ENVIRONMENT

- An Android Virtual Device (AVD) allows developers to test their applications without hooking up an actual Android phone.
- AVDs can be created in various configurations to emulate different types of real phones

PACKAGING IN ANDROID

- The final static component of an application is, of course, the packaged application itself.
- Android provides an application called apkbuilder for generating installable Android application files, which have the extension *.apk*.
- An *.apk* file is in ZIP file format, just like many other Java-oriented application formats, and contains the application manifest, compiled application classes, and all of the resources.

.APK FILE

- Once the *.apk* file has been created, it can be installed onto a device (or emulator) in one of several ways:
  - Using the *adb* interface directory, or more commonly by using an IDE
  - Using an SD card
  - Making the file available on a web server
  - Uploading the file to the Android Market, and then selecting Install