AppDev

1 Android App Components

□ What are the four types of app components and what is their purpose?

* Activities
* Services
* Broadcast receivers
* Content providers

**Activities**

An activity is the entry point for interacting with the user. It 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 to allow the user to share a picture. An activity facilitates the following key interactions between system and app:

* Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
* Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
* Helping the app handle having its process killed so the user can return to activities with their previous state restored.
* Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

You implement an activity as a subclass of the Activity class. For more information about the Activity class, see the Activities developer guide.

**Services**

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It 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.

There are two types of services that tell the system how to manage an app: started services and bound services.

**Started services** tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represent two different types of started services that modify how the system handles them:

* Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
* A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.
* Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about.
* Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running.
* A service is implemented as a subclass of [Service](https://developer.android.com/reference/android/app/Service). For more information about the [Service](https://developer.android.com/reference/android/app/Service) class, see the Services developer guide.

**Broadcast receivers**

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event... and by delivering that alarm to a BroadcastReceiver of the app, there is no need for the app to remain running until the alarm goes off. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may [create a status bar notification](https://developer.android.com/guide/topics/ui/notifiers/notifications) to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. For instance, it might schedule a [JobService](https://developer.android.com/reference/android/app/job/JobService) to perform some work based on the event with [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler)

A broadcast receiver is implemented as a subclass of [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver) and each broadcast is delivered as an [Intent](https://developer.android.com/reference/android/content/Intent) object. For more information, see the [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver) class.

**Content providers**

A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or 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 the content provider, such as [ContactsContract.Data](https://developer.android.com/reference/android/provider/ContactsContract.Data), to read and write information about a particular person. It is tempting to think of a content provider as an abstraction on a database, because there is a lot of API and support built in to them for that common case. However, they have a different core purpose from a system-design perspective. To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme. Thus an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities which can in turn use them to access the data. There are a few particular things this allows the system to do in managing an app:

* Assigning a URI doesn't require that the app remain running, so URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.
* These URIs also provide an important fine-grained security model. For example, an app can place the URI for an image it has on the clipboard, but leave its content provider locked up so that other apps cannot freely access it. When a second app attempts to access that URI on the clipboard, the system can allow that app to access the data via a temporary *URI permission grant* so that it is allowed to access the data only behind that URI, but nothing else in the second app.

Content providers are also useful for reading and writing data that is private to your app and not shared.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

□ Does an app need to define one unique entry point or can it have multiple ones? How does the underlying paradigm differ from desktop applications?

You don’t have to define the entry points explicitly. It can have multiple ones. Every activity, service and contentProvider is a entry point.

□ What is the difference between explicit and implicit Intents?

An intent is created with an [Intent](https://developer.android.com/reference/android/content/Intent) object, which defines a message to activate either a specific component (explicit intent) or a specific type of component (implicit intent).

□ What is the "application context" and what is the role of the class "R"?

Is a combination of application manifest file and the resources.  
 May be used in the application code to gain access to the resources at runtime.  
  
 Class R is to connect the MainActivity with buttons or text from activity\_main.xml.