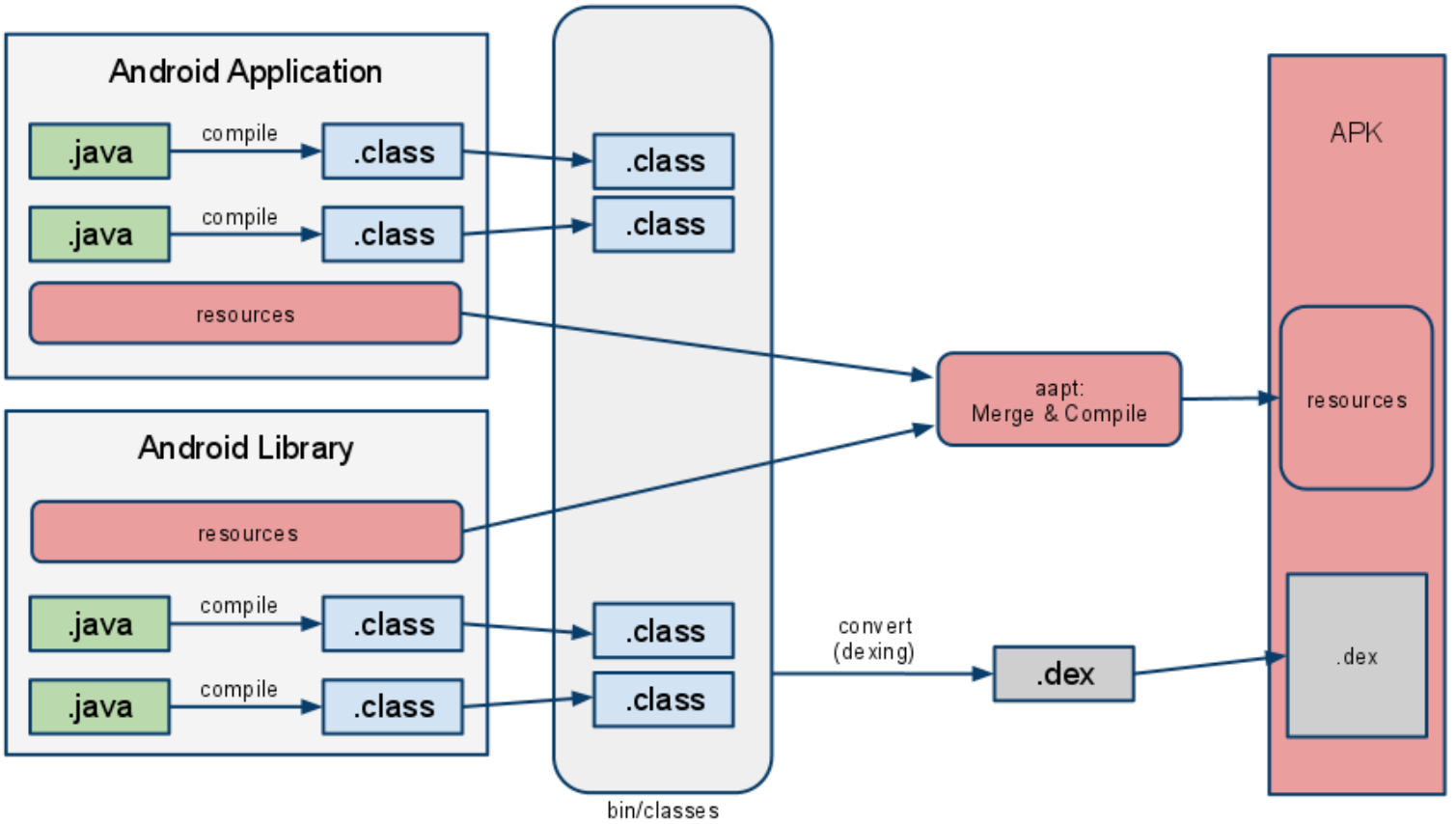
Application Fundamentals

Android apps can be written using Kotlin, Java, and C++ languages. The Android SDK tools compile your code along with any data and resource files into an APK, an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

Application Fundamentals



Application Security

Each Android app lives in its own security **sandbox**, protected by the following Android security features:

- The Android operating system is a multi-user Linux system in which each app is a different user.
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

App components

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. The following sections describe the four types of app components.

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:

Activities Purpose

- 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.
- 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.

Services States

A service can essentially take two states −

- **Started service**:- 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.

Services States

- **Bound Services:-** A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

  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.

Choosing between a service and a thread

A service is simply a component that can run in the background, even when the user is not interacting with your application, so you should create a service only if that is what you need.

If you must perform work outside of your main thread, but only while the user is interacting with your application, you should instead create a new thread. For example, if you want to play some music, but only while your activity is running, you might create a thread in onCreate(), start running it in onStart(), and stop it in onStop().

Broadcast receivers

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow. 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...

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, to read and write information about a particular person.

Example

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.

Activating components

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime. You can think of them as the messengers that request an action from other components, whether the component belongs to your app or another.An intent is created with an Intent object, which defines a message to activate either a specific component (explicit intent) or a specific type of component (implicit intent).

Example

- 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).
- You can initiate a broadcast by passing an Intent to methods such as sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
- You can perform a query to a content provider by calling query() on a ContentResolver.

The manifest file

Before the Android system can start an app component, the system must know that the component exists by reading the app's manifest file, AndroidManifest.xml. Your app must declare all its components in this file, which must be at the root of the app project directory.

- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.

Declaring components

- The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an activity as follows:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Declaring components

- In the <application> element, the android:icon attribute points to resources for an icon that identifies the app.

  In the <activity> element, the android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attribute specifies a string to use as the user-visible label for the activity.
- <activity> elements for activities.
- <service> elements for services.
- <receiver> elements for broadcast receivers.
- <provider> elements for content providers.

Intents and Intent Filters

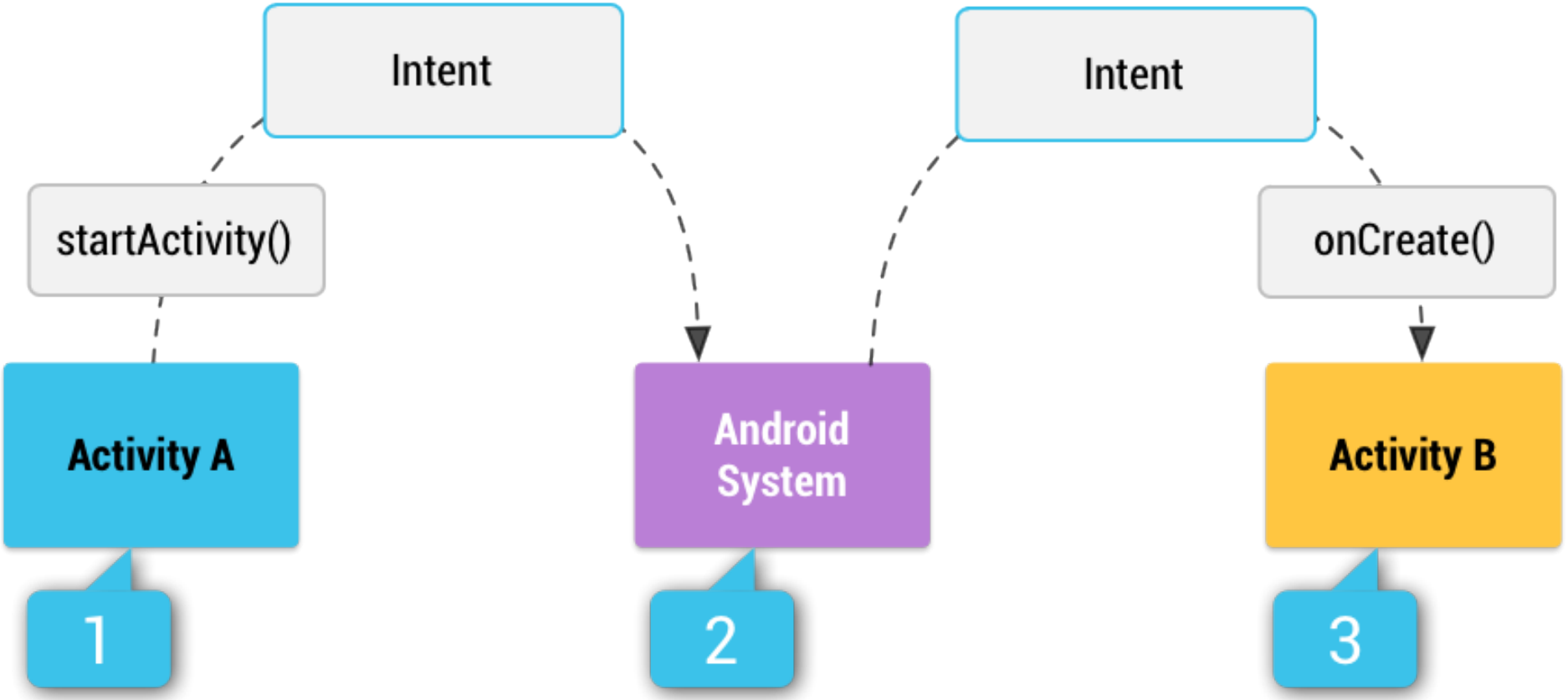
An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases:

- Starting an activity
- Starting a service
- Delivering a broadcast

Intent types

- **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Intents works



Building an intent

An Intent object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

Building an intent

The primary information contained in an Intent is the following:

- **Component name**
- **Action**
- **Data**
- **Category**
- **Extras**
- **Flags**

Component name

The name of the component to start.

This is optional, but it's the critical piece of information that makes an intent explicit, meaning that the intent should be delivered only to the app component defined by the component name. Without a component name, the intent is implicit and the system decides which component should receive the intent based on the other intent information (such as the action, data, and category—described below).

Action

A string that specifies the generic action to perform (such as view or pick).

You can specify your own actions for use by intents within your app (or for use by other apps to invoke components in your app), but you usually specify action constants defined by the Intent class or other framework classes.

e.g **ACTION_VIEW**, **ACTION_SEND**

Example

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.google.com/"));
startActivity(i);
```

```
Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
photoPickerIntent.setType("image/*");
startActivityForResult(photoPickerIntent, SELECT_PHOTO);
```

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to Send");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

```
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Data

- The URI (a Uri object) that references the data to be acted on and/or the MIME type of that data.

  When creating an intent, it's often important to specify the type of data (its MIME type) in addition to its URI. For example, an activity that's able to display images probably won't be able to play an audio file, even though the URI formats could be similar. Specifying the MIME type of your data helps the Android system find the best component to receive your intent.

Category

A string containing additional information about the kind of component that should handle the intent. Any number of category descriptions can be placed in an intent, but most intents do not require a category. Here are some common categories:

e.g **CATEGORY_BROWSABLE** , **CATEGORY_LAUNCHER**

Extras

Key-value pairs that carry additional information required to accomplish the requested action. ou can add extra data with various putExtra() methods, each accepting two parameters: the key name and the value. You can also create a Bundle object with all the extra data, then insert the Bundle in the Intent with putExtras().