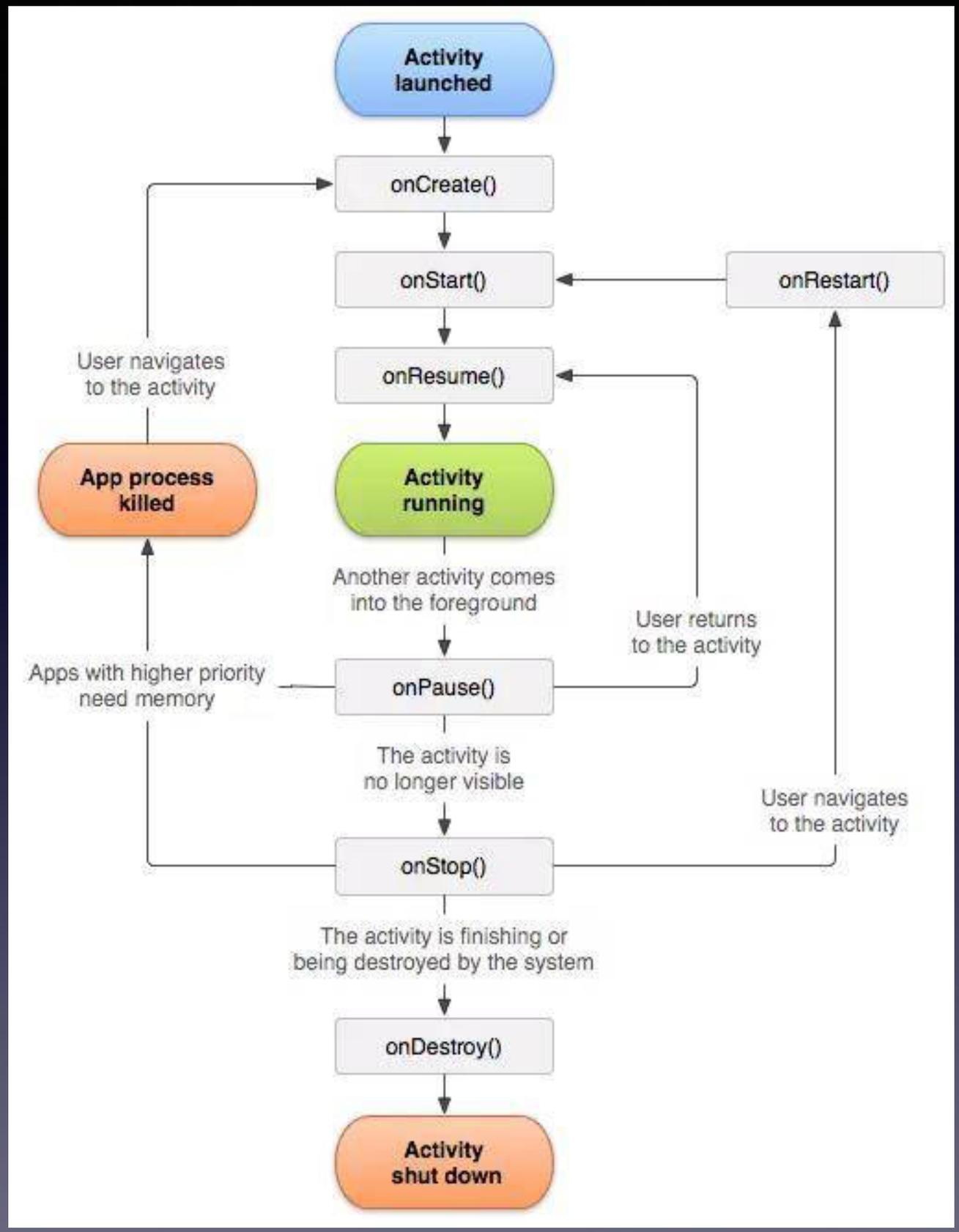
Introduction to Mobile Programming

Android Programming

Chapter 1

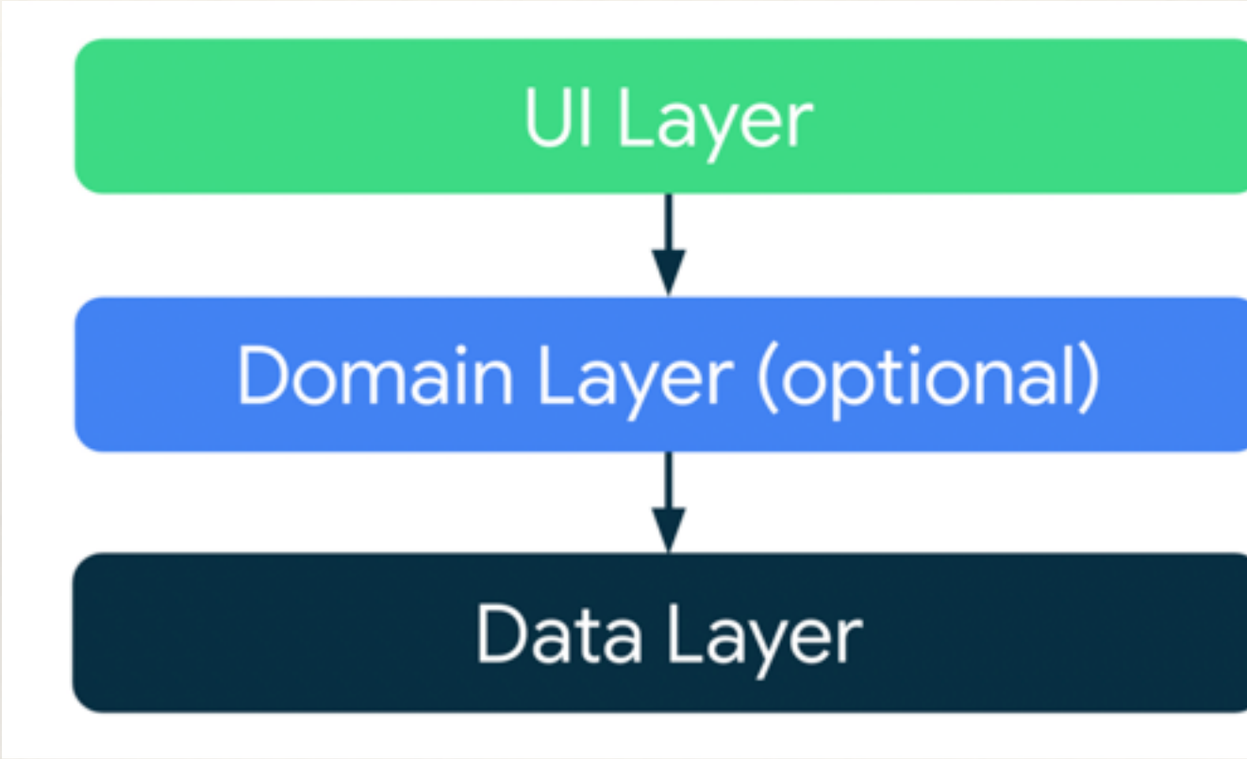
Android Architecture



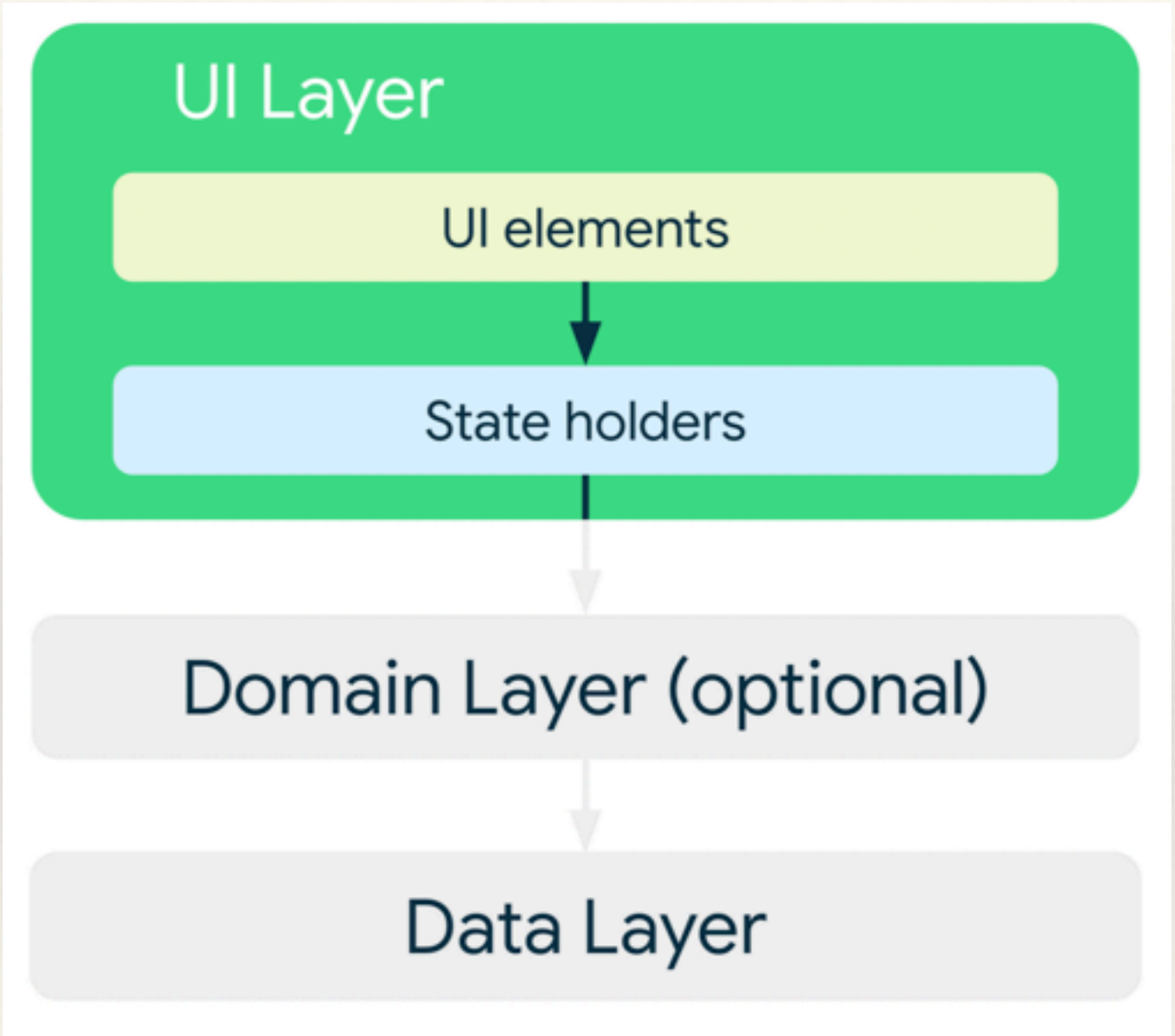
Android Lifecycle

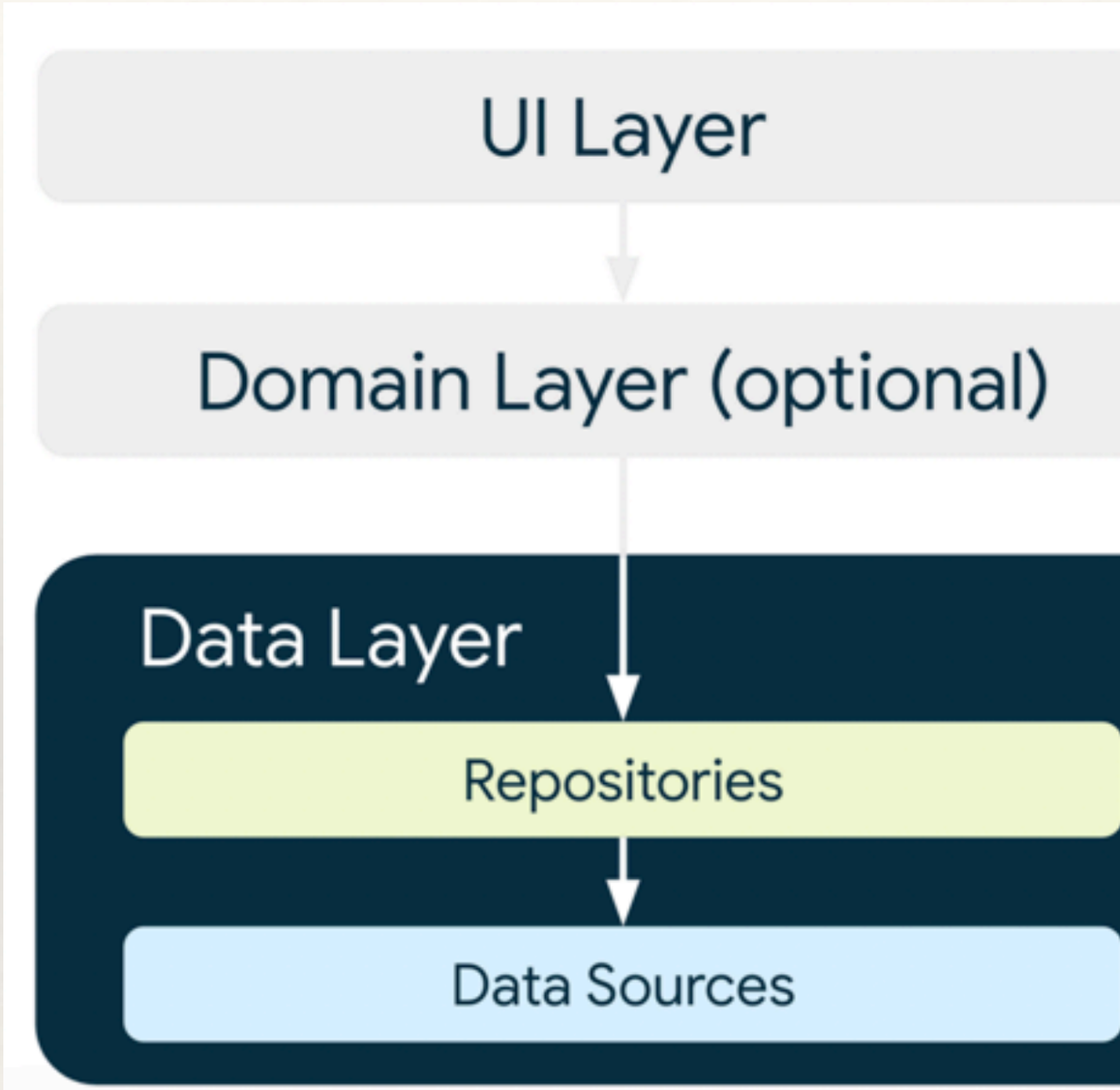
Common Architectural Principles

- Seperation of concerns
- Drive UI from data models
- Single source of truth
- Undirectional Data Flow



Modern App Architecture





General Best Practices

- Do not store data in app components
- Reduce dependencies on Android classes
- Create well-defined boundaries of responsibility between various modules in your app
- Expose as little as possible from each module.
- Focus on the unique core of your app so it stands out from other apps
- Consider how to make each part of your app testable in isolation
- Types are responsible for their concurrency policy.
- Persist as much relevant and fresh data as possible.

Benefits of Architecture

- It improves the maintainability, quality and robustness of the overall app.
- It allows the app to scale. More people and more teams can contribute to the same codebase with minimal co conflicts.
- It helps with onboarding. As Architecture brings consistency to your project, new members of the team can qu get up to speed and be more efficient in less amount of time.
- It is easier to test. A good Architecture encourages simpler types which are generally easier to test.
- Bugs can be investigated methodically with well defined processes.

Build Your First App

- Apps provide multiple entry points
  - Components
    - Activity, BroadcastReceivers, Services, …
    - App Icon, Different Activity, Notification, Different App
- Apps adapt to different devices
  - Different layouts for different screen sizes
  - Specific hardware

Application Fundamentals

- Kotlin, Java, C++
- Code and resource files are compiled into APK
  - APK - an archive file, an Android Package
    - An *Android package*, which is an archive file with an `.apk` suffix, contains the contents of an Android app that are required at runtime and it is the file that Android-powered devices use to install the app.
- One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app
- An Android App Bundle, which is an archive file with an `.aab` suffix, contains the contents of an Android app project including some additional metadata that is not required at runtime

Application Fundamentals

- Each Android app lives in its own security sandbox
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- The Android system implements the *principle of least privilege*.

Principle of Least Privilege

- Each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission. However, there are ways for an app to share data with other apps and for an app to access system services.

- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. The apps must also be signed with the same certificate.

- An app can request permission to access device data such as the device's location, camera, and Bluetooth connection. The user has to explicitly grant these permissions.

Android Programming

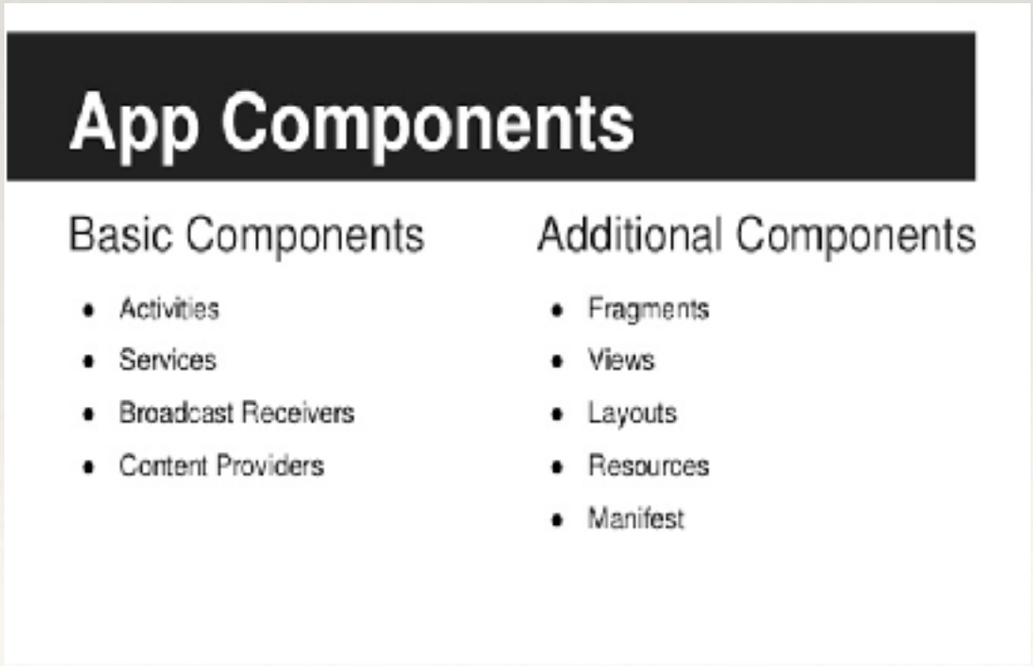
App Components

- App components are the essential building blocks of an Android app.

- Essential Building Blocks
    - Activities
    - Services
    - Broadcast Receivers
    - Content Providers

- AndroidManifest file

- App Resources



Android Programming

Activities

- An *activity* is the entry point for interacting with the user.
- It represents a single screen with a user interface.
- An e-mail app

Android Programming

Services

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

The service may allow the system 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.

Android Programming

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

- Many broadcasts originate from the system.

- Apps can also initiate broadcasts.

- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs

Android Programming

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.

- Any app with the proper permissions can query the content provider

Android Programming

Intents - Activating Components

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*.

- An intent defines the action to perform (for example, to *view* or *send* something) and may specify the URI of the data to act on, among other things that the component being started might need to know.

Android Programming

The 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
- Declares the minimum API Level required by the app
- Declares hardware and software features used or required by the app
- Declares API libraries the app needs to be linked against

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Android Programming

App Resources

- Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.

App Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>
    <string name="hi">@string/hello</string>
</resources>
```

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#f00</color>
    <color name="highlight">@color/red</color>
</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/secondary_text_dark"
    android:text="@string/hello" />
```

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

Android Programming

Device Configuration Changes

- Some device configurations can change during runtime (such as screen orientation, keyboard availability, and when the user enables multi-window mode).

- onSaveInstanceState()

Android Programming

App Permissions

Permission Approval

- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>
</manifest>
```

Android Programming

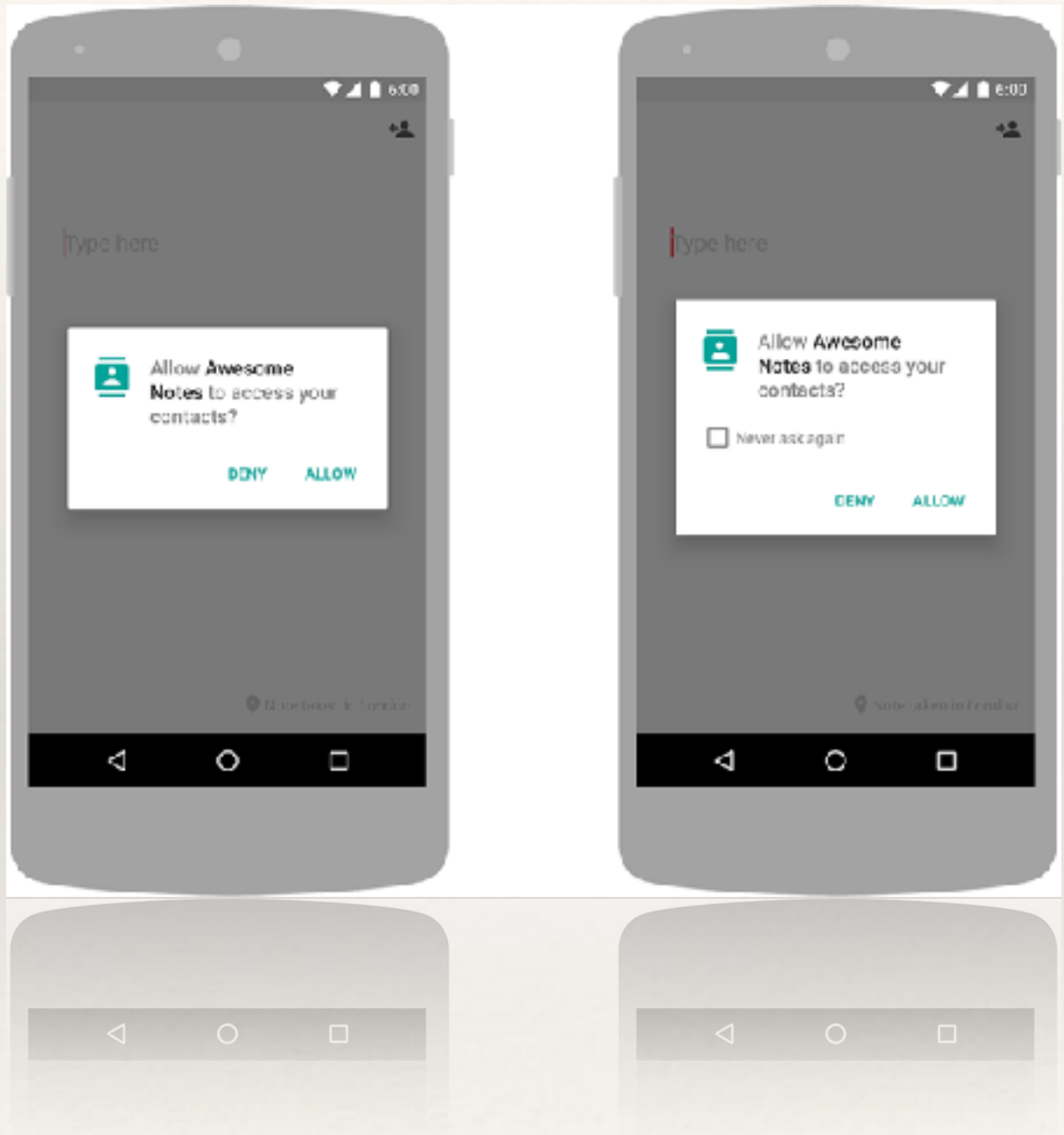
App Permissions

- Normal Permissions
    - permissions that don't pose much risk to the user's privacy or the device's operation

- Dangerous Permissions
    - permissions that could potentially affect the user's privacy or the device's normal operation

Android Programming

Runtime Requests

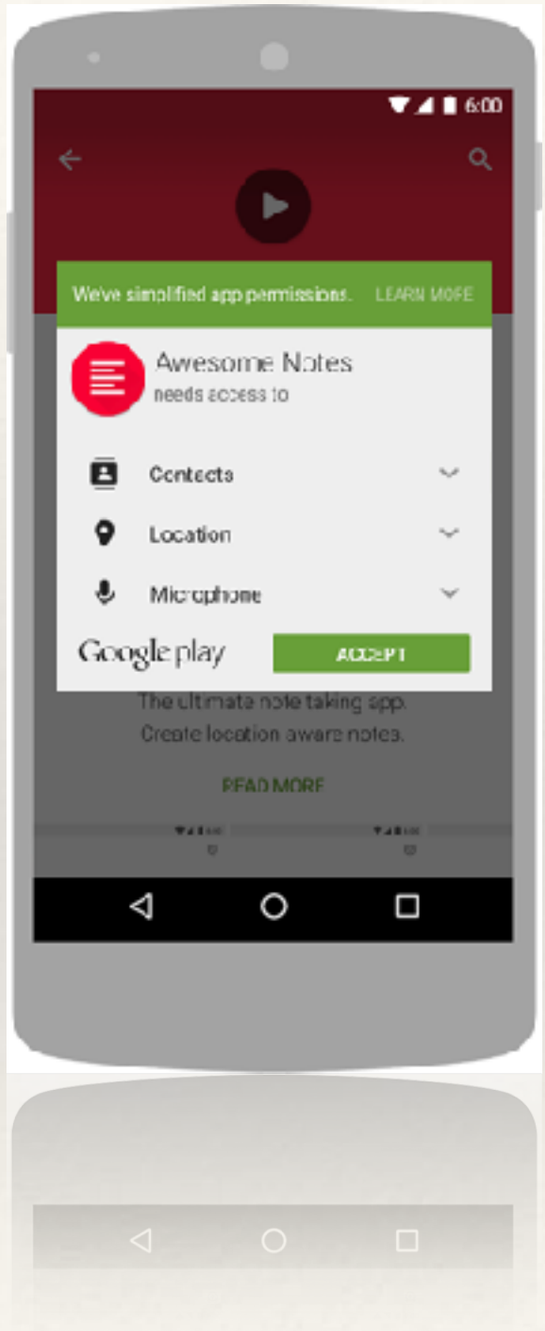


- If the device is running Android 6.0 (API level 23) or higher, *and* the app's `targetSdkVersion` is 23 or higher, the user isn't notified of any app permissions at install time.

- Your app must ask the user to grant the dangerous permissions at runtime.
-

Android Programming

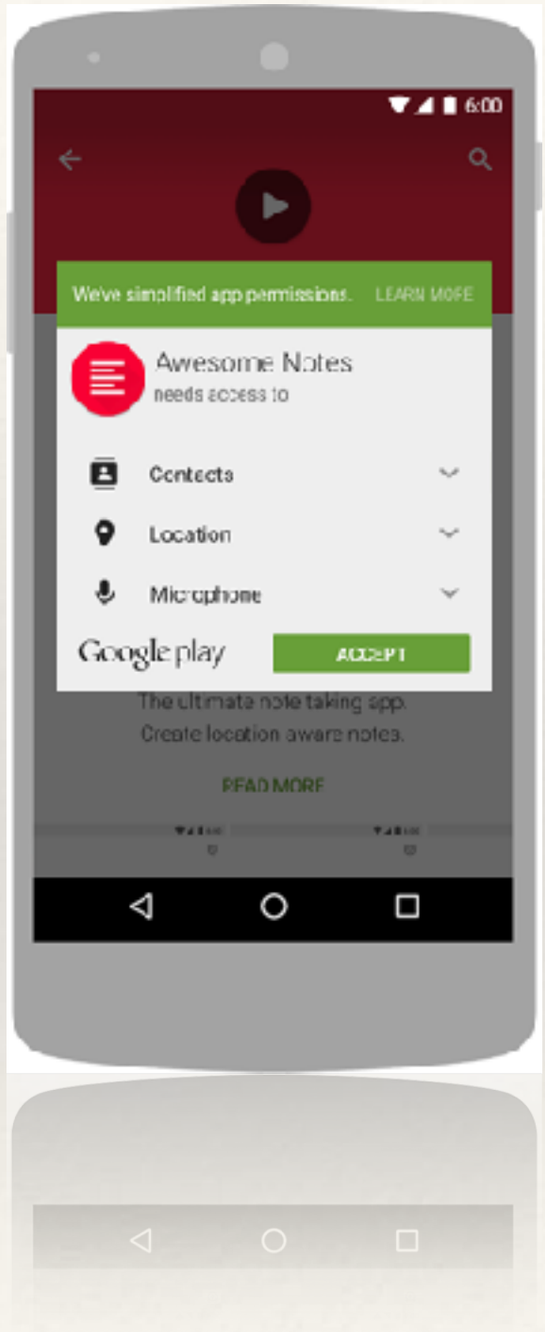
Install-time Requests



- If the device is running Android 5.1.1 (API level 22) or lower, *or* the app's `targetSdkVersion` is 22 or lower while running on any version of Android, the system automatically asks the user to grant all dangerous permissions for your app at install-time
-

Android Programming

Permissions for Optional Hardware Features



- Access to some hardware features (such as Bluetooth or the camera) require an app permission. However, not all Android devices actually have these hardware features.

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

Normal Permissions

- As of Android 9 (API level 28), the following permissions are classified as `PROTECTION_NORMAL`

- `ACCESS_LOCATION_EXTRA_COMMANDS`
- `ACCESS_NETWORK_STATE`
- `ACCESS_NOTIFICATION_POLICY`
- `ACCESS_WIFI_STATE`
- `BLUETOOTH`
- `BLUETOOTH_ADMIN`
- `BROADCAST_STICKY`
- `CHANGE_NETWORK_STATE`
- `CHANGE_WIFI_MULTICAST_STATE`
- `CHANGE_WIFI_STATE`
- `DISABLE_KEYGUARD`
- `EXPAND_STATUS_BAR`
- `FOREGROUND_SERVICE`
- `GET_PACKAGE_SIZE`
- `INSTALL_SHORTCUT`
- `INTERNET`
- `KILL_BACKGROUND_PROCESSES`
- `MANAGE_OWN_CALLS`
- `MODIFY_AUDIO_SETTINGS`
- `NFC`
- `READ_SYNC_SETTINGS`
- `READ_SYNC_STATS`
- `RECEIVE_BOOT_COMPLETED`
- `REORDER_TASKS`
- `REQUEST_COMPANION_RUN_IN_BACKGROUND`
- `REQUEST_COMPANION_USE_DATA_IN_BACKGROUND`
- `REQUEST_DELETE_PACKAGES`
- `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS`
- `SET_ALARM`
- `SET_WALLPAPER`
- `SET_WALLPAPER_HINTS`
- `TRANSMIT_IR`
- `USE_FINGERPRINT`
- `VIBRATE`
- `WAKE_LOCK`
- `WRITE_SYNC_SETTINGS`

Signature Permissions

- The system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission.

- `BIND_ACCESSIBILITY_SERVICE`
- `BIND_AUTOFILL_SERVICE`
- `BIND_CARRIER_SERVICES`
- `BIND_CHOOSER_TARGET_SERVICE`
- `BIND_CONDITION_PROVIDER_SERVICE`
- `BIND_DEVICE_ADMIN`
- `BIND_DREAM_SERVICE`
- `BIND_INCALL_SERVICE`
- `BIND_INPUT_METHOD`
- `BIND_MIDI_DEVICE_SERVICE`
- `BIND_NFC_SERVICE`
- `BIND_NOTIFICATION_LISTENER_SERVICE`
- `BIND_PRINT_SERVICE`
- `BIND_SCREENING_SERVICE`
- `BIND_TELECOM_CONNECTION_SERVICE`
- `BIND_TEXT_SERVICE`
- `BIND_TV_INPUT`
- `BIND_VISUAL_VOICEMAIL_SERVICE`
- `BIND_VOICE_INTERACTION`
- `BIND_VPN_SERVICE`
- `BIND_VR_LISTENER_SERVICE`
- `BIND_WALLPAPER`
- `CLEAR_APP_CACHE`
- `MANAGE_DOCUMENTS`
- `READ_VOICEMAIL`
- `REQUEST_INSTALL_PACKAGES`
- `SYSTEM_ALERT_WINDOW`
- `WRITE_SETTINGS`
- `WRITE_VOICEMAIL`

Dangerous Permissions

-

Table 1. Dangerous permissions and permission groups.

Permission Group	Permissions
CALENDAR	• READ_CALENDAR • WRITE_CALENDAR
CALL_LOG	• READ_CALL_LOG • WRITE_CALL_LOG • PROCESS_OUTGOING_CALLS
CAMERA	• CAMERA
CONTACTS	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
LOCATION	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
MICROPHONE	• RECORD_AUDIO
PHONE	• READ_PHONE_STATE • READ_PHONE_NUMBERS • CALL_PHONE • ANSWER_PHONE_CALLS • ADD_VOICEMAIL • USE_SIP
SENSORS	• BODY_SENSORS
SMS	• SEND_SMS • RECEIVE_SMS • READ_SMS • RECEIVE_WAP_PUSH • RECEIVE_MMS
STORAGE	• READ_EXTERNAL_STORAGE • WRITE_EXTERNAL_STORAGE

Request App Permissions

- Add permissions to the manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.snazzyapp">

   <uses-permission android:name="android.permission.INTERNET"/>
   <!-- other permissions go here -->

   <application ...>
      ...
   </application>
</manifest>
```

- Check for permissions
  - If your app needs a dangerous permission, you must check whether you have that permission every time you perform an operation that requires that permission.

  - ContextCompat.checkSelfPermission()
    - PERMISSION_GRANTED
    - PERMISSION_DENIED
  - requestPermissions()

Android Programming

Create an Android Project

-Android Studio
-Create New Project
-Application Name
-Company Domain
-Empty Activity
-MainActivity
-Layout file
-AndroidManifest.xml
-Gradle Scripts > build.gradle

Android Programming

Run Your App

- Run on a real device
  Enable USB debugging in the Developer options

- Run on an emulator
  Create New Virtual Device

Android Programming

Build a simple user interface

- The user interface for an Android app is built using a hierarchy of
  - -layouts
  - -widgets

- Open the Layout Editor
  - - Select an Layout
  - - Add a text box
  - - Add a button
  - - Change the UI strings