Introduction to Android
Fundemental Concepts

Çağatay Sönmez

03.02.2021

Arcelik Electronics Plant, R&D Team - Software Design

Agenda

- What is Android
- Manifest File
- Activities
- Intents
- Services
- Threading in Android
- Broadcasts
- Content Providers
- Permissions
- Android Storage

What is Android

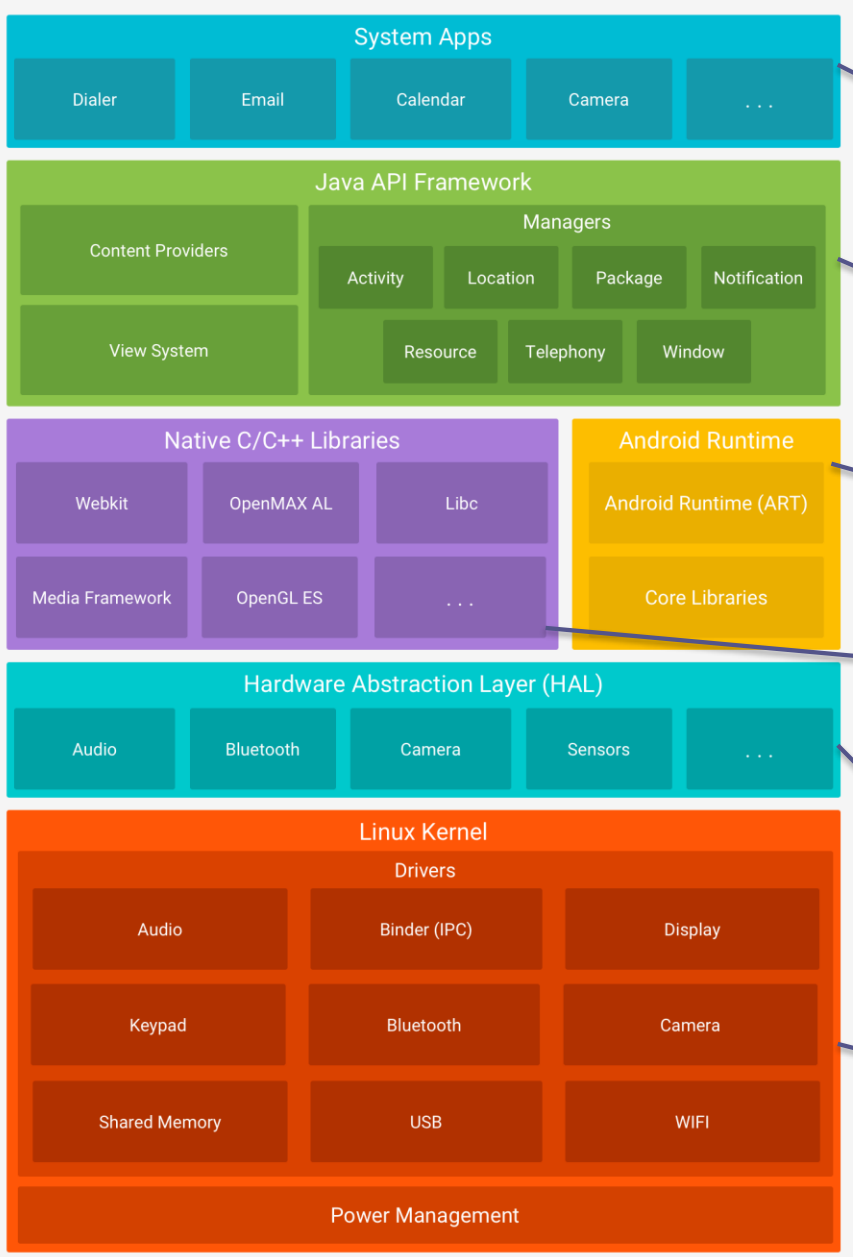
- Android is a Linux based operating system
- It has modified version of the Linux kernel including other open source software
- Initially developed by Android Inc in 2003
- Google bought Android Inc. in 2005
- Primarily designed for touchscreen mobile devices
- An open source platform, Google is the principle maintainer
- Source code is available:
  - https://android.googlesource.com

Android Versions

Code Name	Version	Linux kernel	Release
(No codename)	1.0	?	2008
Petit Four	1.1	2.6	2009
Cupcake	1.5	2.6.27	2009
Donut	1.6	2.6.29	2009
Éclair	2.0 – 2.1	2.6.29	2009
Froyo	2.2 – 2.2.3	2.6.32	2010
Gingerbread	2.3 – 2.3.7	2.6.35	2010
Honeycomb	3.0 – 3.2.6	2.6.36	2011
Ice Cream Sandwich	4.0 – 4.0.4	3.0.1	2011
Jelly Bean	4.1 – 4.3.1	3.0.31 to 3.4.39	2012
KitKat	4.4 – 4.4.4	3.10	2013
Lollipop	5.0 – 5.1.1	3.16	2014
Marshmallow	6.0 – 6.0.1	3.18	2015
Nougat	7.0 – 7.1.2	4.4	2016
Oreo	8.0 – 8.1	4.10	2017
Pie	9.0	4.4 to 4.14	2018
Q	10	4.9 to 4.19	2019
R	11	4.14 to 5.14	2020

Android Development Tools

- Android Studio
  - The official IDE for Android app development
- Android SDK Tools (comes with Android Studio)
  - adb
  - emulator
  - lint
  - ...
- Android NDK
  - Lets us use C/C++ codes in Android app



set of core apps including the 3rd party apps

feature-set of the APIs written in the Java language.

ART replaces Dalvik Virtual Machine prior to Android 5.0

Many core Android system components written in C/C++

standard interfaces to expose hw capabilities to the higher-level

The foundation of the Android platform is the Linux kernel.

Android Core Building Blocks

AndroidManifest.xml

- Every app project must have an AndroidManifest.xml file
- Corresponding XML element must be declared for each app component
  - <activity> for each subclass of Activity
  - <service> for each subclass of Service
  - <receiver> for each subclass of BroadcastReceiver
  - <provider> for each subclass of ContentProvider
- Intent filters and Permissions must be declared in manifest file
- Minimum and target SDK version can be declared in manifest file
- Check below link for more
  - https://developer.android.com/guide/topics/manifest/manifest-intro

AndroidManifest.xml Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

       <activity android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />

</manifest>
```

Activity

- The mobile-app experience is different than the desktop app
  - User's interaction doesn't always begin in the same place
  - An app can invoke a specific window of the other app
- An activity provides the window in which the app draws its UI
- Generally, one activity implements one screen in an app
- Each activity can then start another activity in order to perform different actions
- Each activity is loosely bound to the other activities; there are usually minimal dependencies among the activities in an app

Activity - Configuring the manifest

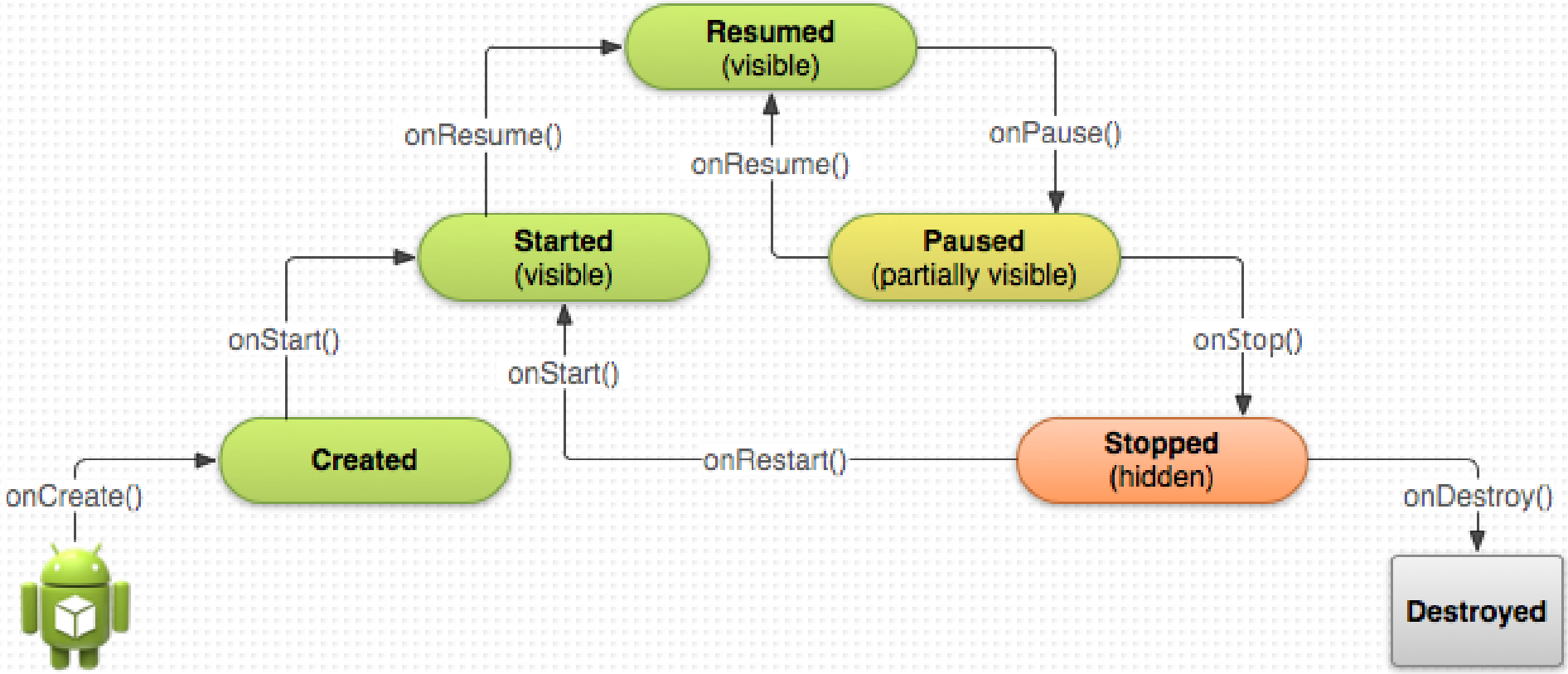
- You must declare activities and their attributes in manifest.xml
- The only required attribute is **android:name** (the class name)
- There are many attributes for the activity element
  - android:theme
  - android:noHistory
  - android:permission
  - android:screenOrientation
  - …
- Check below link for whole attributes
  - https://developer.android.com/guide/topics/manifest/activity-element

Starting Activity

- An activity often needs to start another activity at some point
- The new activity is started using either
  - startActivity() method
  - startActivityForResult() method

```
…
Intent intent = new Intent(this, SampleActivity.class);
startActivity(intent);
…
```

Activity Life Cycle



Activity – onCreate()

- onCreate() is invoked when the system first creates the activity
- Activity does not reside in the Created state, it quickly moves on to the Started and Resumed states
- After onCreate() method executed, the system calls the onStart() and onResume() methods in quick succession
- Commonly used to initialize and configure UI

Activity – onStart()

- onStart() is invoked when the activity enters the Started state
- The onStart() call makes the activity visible to the user
- The onStart() method completes very quickly
- Once this callback finishes, the system invokes onResume()
- Commonly used to check if required system features are enabled

Activity – onResume()

- onResume callback is invoked when the Activity is completely visible, focused and ready for user interaction.
- The app stays in this state until something happens to take focus away from the app
- Commonly used to handle CPU intensive actions

Activity – onPause()

- onPause callback is called when the user is leaving the activity, for example, the user taps the Back or Recents button
- When it is called the activity is no longer in the foreground
- It may still be visible if the user is in multi-window mode or a new semi-transparent activity (such as a dialog) opens
  - Only one of the apps (windows) has focus at any time, the system pauses all of the other apps
- onPause() execution is very brief, and does not necessarily afford enough time to perform save operations

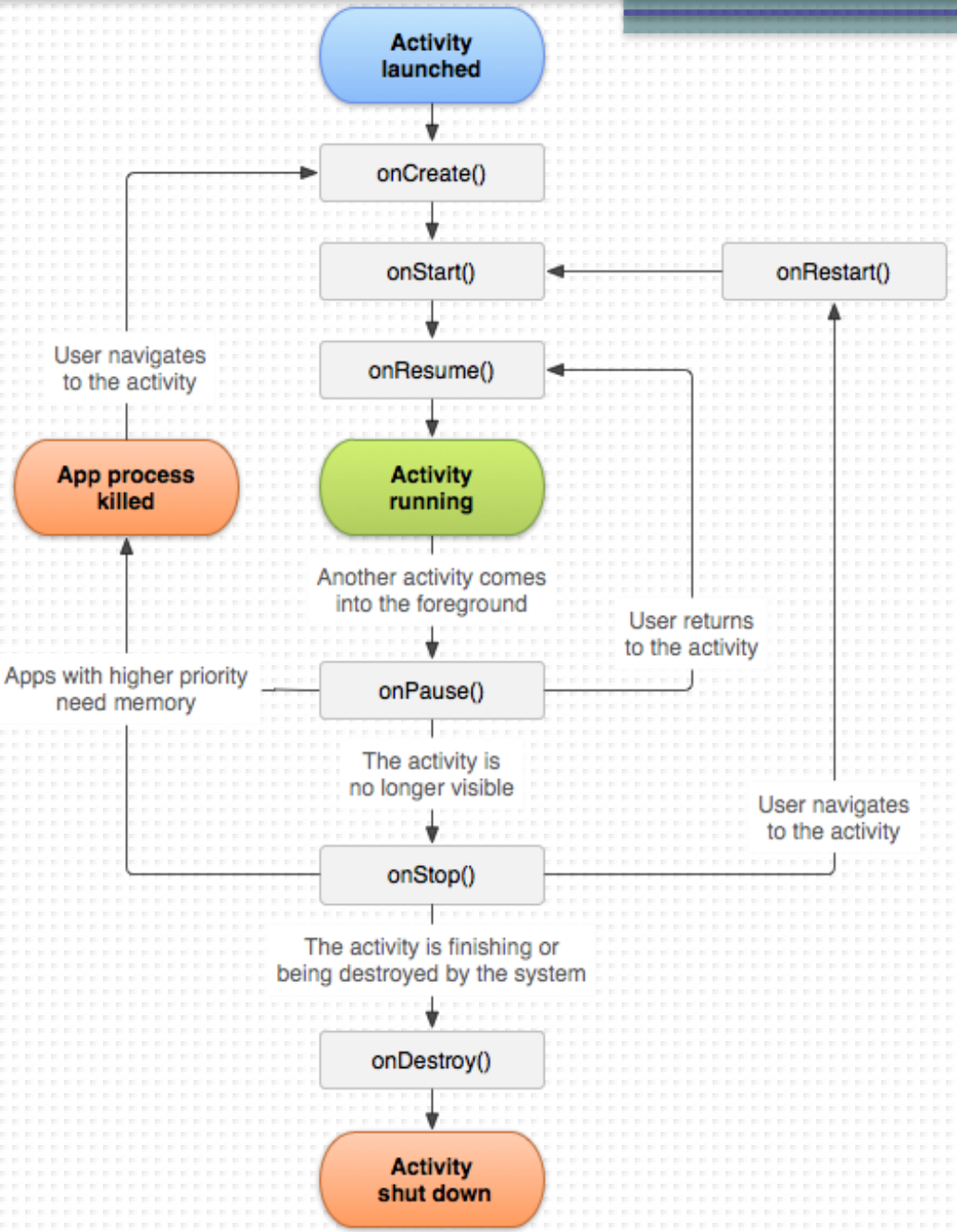
Activity- onStop()

- onStop callback is invoked when the activity is no longer visible
  - When a newly launched activity covers the entire screen
  - When the activity has finished running, and is about to be terminated
- onStop() is the last method in the lifecycle that is guaranteed to be called
  - onDestroy may not be called in low memory situations
- onStop() is commonly used to perform CPU-intensive shutdown operations and release resources that might leak memory

Activity - onDestroy

- onDestroy callback is called before the activity is destroyed.
- The system invokes this callback either because:
  - finish() being called on the activity (programmatically)
  - user completely dismissing the activity
  - system is temporarily destroying the activity due to a configuration change (such as device rotation)
- The system never kills an activity directly, if Android kills the application process, all activities are terminated.

Overview


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

When the System Kills a Process?

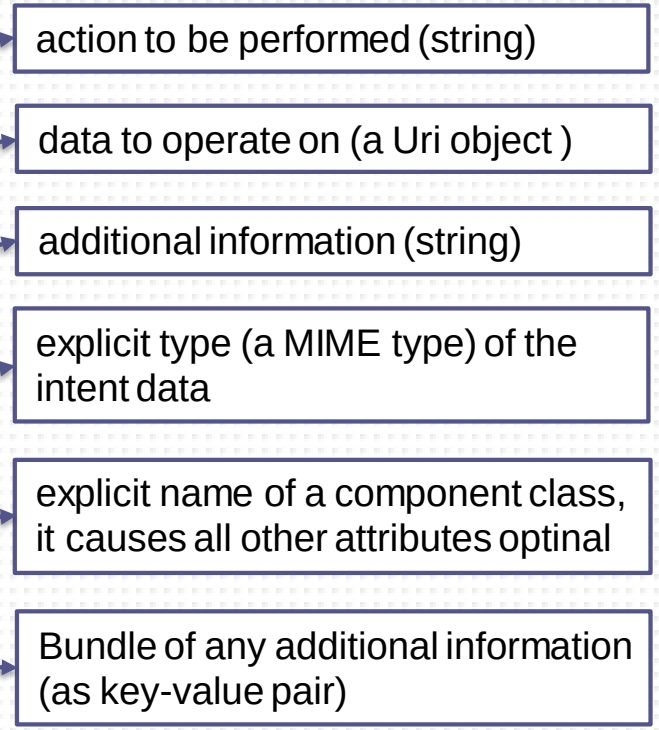
- The system kills processes when it needs to free up RAM
- The likelihood of the system killing a given process depends on the state of the process at the time

Likelihood of being killed	Process state	Activity state
Least	Foreground (having or about to get focus)	Created Started Resumed
More	Background (lost focus)	Paused
Most	Background (not visible)	Stopped
	Empty	Destroyed

* User can also kill a process by using the Application Manager under Settings to kill an app

Intent

- An Intent is a messaging object you can use to request an action from another app component
- Intent is basically a passive data structure holding an abstract description of an action to be performed
- Primary attributes
  - action
  - data
- Secondary attributes
  - category
  - type
  - component
  - extras



Intent Constructors

```
…
public Intent ()
…
public Intent (Intent o)
…
public Intent (String action)
…
public Intent (String action, Uri uri)
…
public Intent (Context packageContext,
               Class<?> cls)
…
public Intent (String action, Uri uri, Context
               packageContext, Class<?> cls)
```

- Create an empty intent
- Copy constructor
- Create an intent with a given action
- Create an intent with a given action and data url
- Create an intent for a specific component
- Create an intent for a specific component with a specified action and data

Intent - Fundamental Use Cases

- Starting an activity
  - A new instance of an Activity can be started by passing an Intent to startActivity()
  - The Intent describes the activity to start and carries any necessary data
- Starting a service
  - A new service to perform can be started by passing an Intent to startService()
  - The Intent describes the service to start and carries any necessary data
- Delivering a broadcast
  - You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast()

Explicit intent

- Explicit Intents have **specified a component** which provides the exact class to be run
- Used in the application itself, common examples:
  - to start new Activities or Services
  - to pass data to other Activities or Services

```
Intent intent = new Intent(MainActivity.this, MyIntentService.class);
intent.setAction("ACTION_SEND_NOTIFICATION");
intent.putExtra("NOTIFICATION_METHOD", "TOAST");
startService(intent);
```

Implicit intent

- Implicit Intents have not specified a component, it only specifies action to be performed
- The system determine which of the available components is best to run for that intent

Intent Filters

- An intent filter declares the types of intents that an activity, service, or broadcast receiver provides
  - must contain <action>; <data> and <category> are optional
  - <action> and <category> elements must have *name* attribute
  - <data> can have one or more attributes (*scheme*, *mimeType* etc)
- Intent filters declared in the manifest file

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <data android:scheme="mailto" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Deep Linking with Intent Filters

- When a clicked link or programmatic request invokes a web URI intent
- If there is only one available app that can handle the URI Android system opens it
- Otherwise, allows the user to select an app from a dialog.

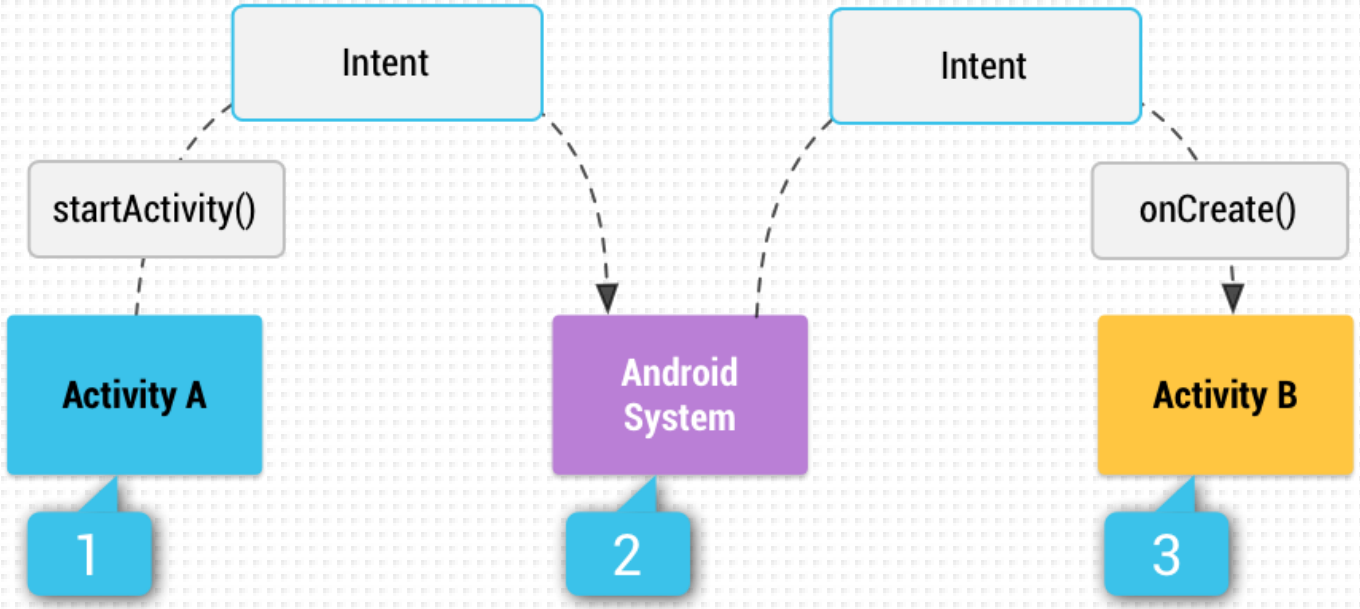
```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <!-- Accepts URIs that begin with "http://www.example.com/demos" -->
    <data android:scheme="http"
          android:host="www.example.com"
          android:pathPrefix="/demos" />
</intent-filter>
```

To receive implicit intent

to be accessible from a web browser

How Implicit Intents are Delivered

- Android System searches all apps for an intent filter
- If multiple apps are found, the system prompt a dialog



Standard Activity Actions

- Standard actions used for launching activities
- ACTION_MAIN, ACTION_VIEW and ACTION_EDIT are commonly used
- Some examples
  - ACTION_CALL
  - ACTION_SEND
  - ACTION_WEB_SEARCH
- See full list below
  - https://developer.android.com/reference/android/content/Intent

Service

- A Service runs in the background to perform long-running operations such as network transaction, playing music
- It doesn't provide a user interface
- A Service is not a separate process or a thread
- The service runs in the background even if app is destroyed
- The system force-stops a service only when memory is low
- A component can bind to a service to interact with it
- Note that the traditional Thread class or AsyncTask can only work while your activity is running

Declaring Service

- To create a service, a subclass of Service must be created
- All services must be declared application's manifest file

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Service Types

- These are the three different types of services
- Foreground service
  - Performs some operation that is noticeable to the user
  - Must display a Notification
- Background service
  - Performs an operation that isn't directly noticed by the user
- Bound service
  - A service is bound when another component binds to it (by calling bindService() method)

Service Callbacks

- Some callback methods that handle key aspects of the service lifecycle must be overridden
- Most important callback methods that should be overridden
  - onStartCommand()
  - onBind()
  - onCreate()
  - onDestroy()

Service - onStartCommand()

- This callback is invoked after another component (such as an activity) calls startService() method.
- When this method executes, the service is started and can run in the background indefinitely.
- If you only want to provide binding, you don't need to implement this method.

```
Intent intent = new Intent(this, MyService.class);
startService(intent);
```

```
public int onStartCommand(Intent intent, int flags, int startId) {
    //do your job here
}
```

Service - onStartCommand Return Value

- The return value describes how the service is handled if the system kills the service after onStartCommand() returns
- The return value must be one of the following constants
- START_NOT_STICKY
  - System does not recreate the service unless there are pending intents to deliver
- START_STICKY
  - System recreates the service and call onStartCommand(), but do not redeliver the last intent
- START_REDELIVER_INTENT
  - System recreates the service and call onStartCommand() with the last intent that was delivered to the service

Service - onBind()

- This callback is invoked after another component (such as an activity) calls bindService() method to bind with the service
- The return value should be an IBinder interface that clients use to communicate with the service
- If you don't want to allow binding, you should return null

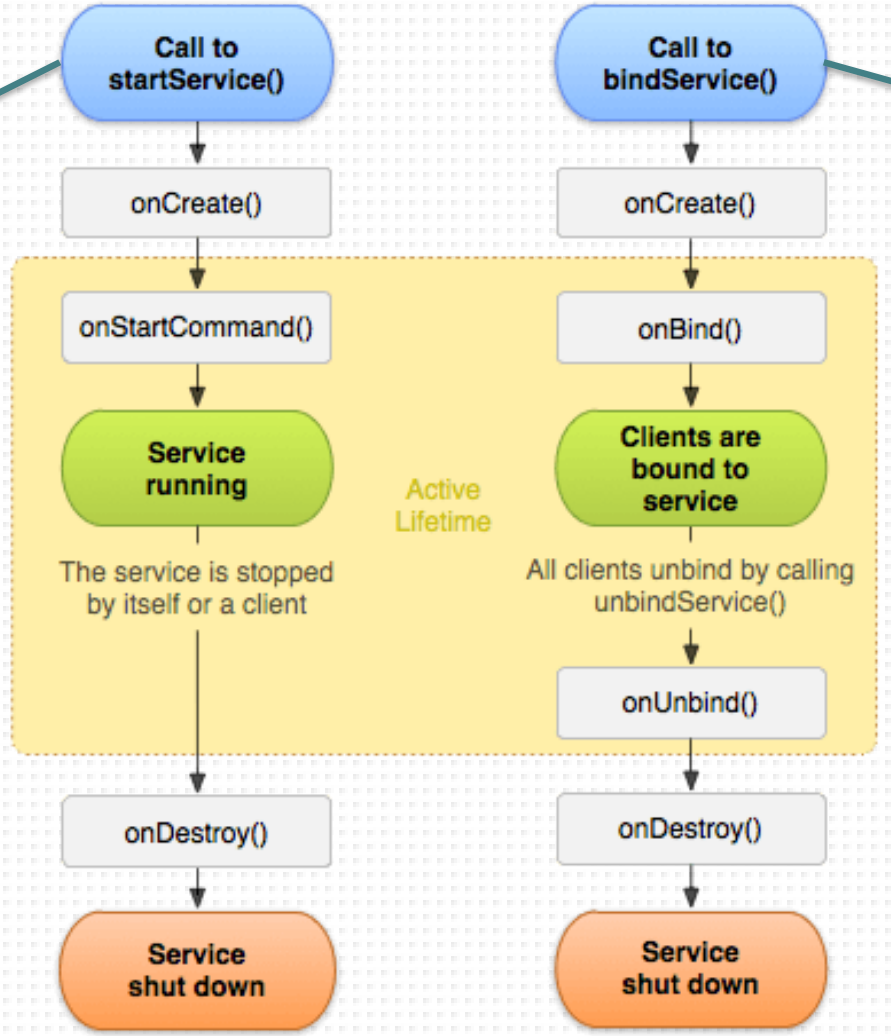
```
public IBinder onBind(Intent intent) {
    // A client is binding to the service with bindService()
    return mBinder;
}
public boolean onUnbind(Intent intent) {
    // All clients have unbound with unbindService()
    return mAllowRebind;
}
public void onRebind(Intent intent) {
    // A client is binding to the service with bindService(),
    // after onUnbind() has already been called
}
```

Service - onCreate() & onDestroy()

- onCreate() callback is invoked when the service is initially created (before onStartCommand() or onBind()) to perform one-time setup procedures
- If the service is already running, onCreate() is not called
- onDestroy() callback is invoked when the service is no longer used and is being destroyed
- onDestroy() is the last call that the service receives
- onDestroy() is used to clean up the resources such as threads, registered listeners, or receivers

Service Life Cycle

Starting unbounded service



Starting bounded service

Service Classes

- There are two classes that can be extended
  - **Service**
  - **IntentService**
- The Service and IntentService can be triggered from any thread, activity or other application component

Service Class

- Runs on background but uses the application's main thread
- May block the Main Thread, so that it can slow the performance of activities
  - **It's important to create a new thread for long tasks!**
- It is developer's responsibility to stop the service
  - **by calling stopSelf() or stopService() methods!**
- Mostly used for multi-threading

IntentService Class

- Uses a worker thread to handle all of the start requests separate from the application's main thread
- **onHandleIntent()** callback should be overridden
  - When this method returns, IntentService stops the service
- Creates a work queue that passes one intent at a time (cannot run tasks in parallel)
  - Don’t have to worry about multi-threading
- Stops after all start requests have been handled
  - Don’t have to call stopSelf()
- Mostly used if simultaneous service request handling is not needed

IntentService Example

```
public class HelloIntentService extends IntentService {
  /**
   * A constructor is required, and must call the super
   * constructor with a name for the worker thread.
   */
  public HelloIntentService() {
      super("HelloIntentService");
  }

  /**
   * The IntentService calls this method from the default worker thread with
   * the intent that started the service
   */
  @Override
  protected void onHandleIntent(Intent intent) {
      // do your work here, like download a file
  }
}
```

Processes on Android

- All components of the same application run in the same process
- When an application is launched, the system creates a thread of execution for the application, called "main"
  - The main thread is also called the UI thread
- Components run in the same process are instantiated in UI thread
  - Methods that respond to system callbacks (such as onKeyDown()) always run in the UI thread
- Android UI toolkit is not thread-safe
  - Do not access the Android UI toolkit from outside the UI thread
- If the UI thread is blocked for more than a few seconds "application not responding" (ANR) dialog is prompted

Threading in Android

- CPU intensive operations should be handled in separate threads because of the single threaded model
- Services can be used to run task in background
  - Services run your task independently of the Activity
- What if you want a thread which should only works when the Activity has focus?
  - Standard Java Thread
  - AsyncTask

Java Thread

- Mostly used for long tasks
- Can be triggered from any thread
- Implement the Runnable interface and pass it to a Thread

```
public class ThreadExample extends Activity {
    …
    Runnable runnable = new Runnable() {
        public void run() {
            //do your work here
        }
    };

    Thread mythread = new Thread(runnable);
    mythread.start();
}
```

Java Thread Handler

- Use handler to pass data to UI thread

```
public class ThreadExample  extends Activity {
    …
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            //do your work with message
        }
    };

    Runnable runnable = new Runnable() {
        public void run() {
            //do your work here
            handler.sendEmptyMessage(0);
        }
    };

    Thread mythread = new Thread(runnable);
    mythread.start();
}
```

Android AsyncTask

- Mostly used for small task having to communicate with UI thread
- Must be triggered from UI thread
- It publishes the results on the UI thread, without requiring you to handle threads
- Implement the **doInBackground()** callback to run background
- Implement **onPostExecute()** callback to safely update UI
  - Delivers the result and runs in the UI thread
- Run the task by calling **execute()** from the UI thread

Android AsyncTask cont.

- Use handle to pass data to UI thread

```
public class AsyncTaskActivity extends Activity {
…
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        new MyTask().execute("dummy parameter");
    }

    private class MyTask extends AsyncTask<String, Void, String> {
        protected String doInBackground(String... params) {
            //do your work here
            return "dummy result";
        }

        protected void onPostExecute(String result) {
            //update UI here
        }
    }
}
```

Threading in Android - BrainStorming

- Which one do you prefer for each scenarios, Service, IntentService, Thread or AsyncTask?
  - Download news feed to display on Activity (UI)
  - Play music while the app is active
  - Track GPS data to push targeted advertising (notification) even if the screen display is off
  - Post app usage statistic data including app close event to server

Broadcasts

- Android apps can send/receive broadcast messages to/from the Android system and other Android apps
- The system automatically routes broadcasts to apps that have subscribed
- The broadcast message itself is wrapped in an Intent object
- Android system automatically sends broadcasts when various system events occur

Sending Broadcasts

- Android provides three ways for apps to send broadcast
  - sendOrderedBroadcast(Intent, String)
  - sendBroadcast(Intent)
  - LocalBroadcastManager.sendBroadcast(Intent)
    - more efficient since no interprocess communication needed

```
Intent intent = new Intent();
intent.setAction("com.example.broadcast.MY_NOTIFICATION");
intent.putExtra("data","Custom notification!");
sendBroadcast(intent);
```

Broadcast Receiver

Specify the <receiver> element in app's manifest

```
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED"  />
    </intent-filter>
</receiver>
```

Implement onReceive(Context, Intent) callback

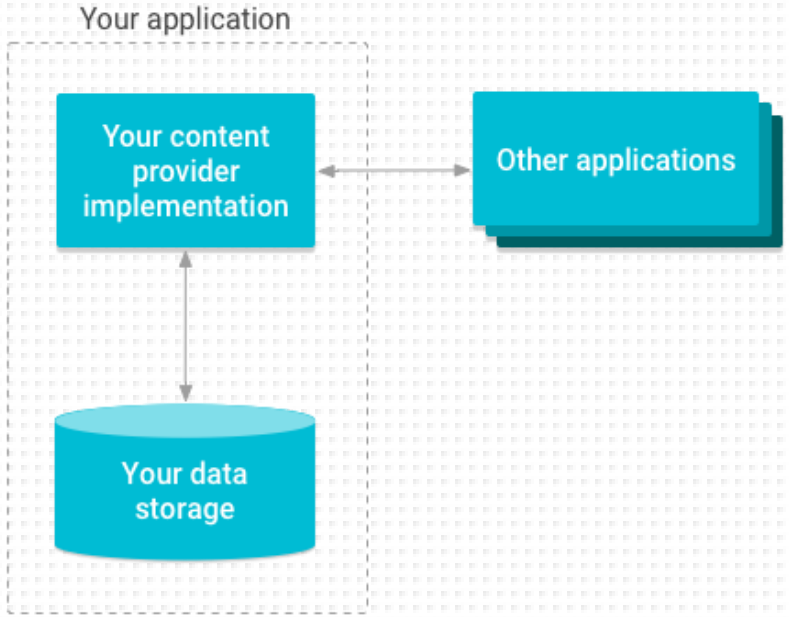
```
public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String log = "Action: " + intent.getAction();
        Toast.makeText(context, log, Toast.LENGTH_LONG).show();
    }
}
```

Standard Broadcast Actions

- Standard broadcast actions used for receiving broadcasts
- Some examples
  - ACTION_TIME_CHANGED
  - ACTION_BOOT_COMPLETED
  - ACTION_POWER_CONNECTED
- See full list below
  - https://developer.android.com/reference/android/content/Intent

Content Providers

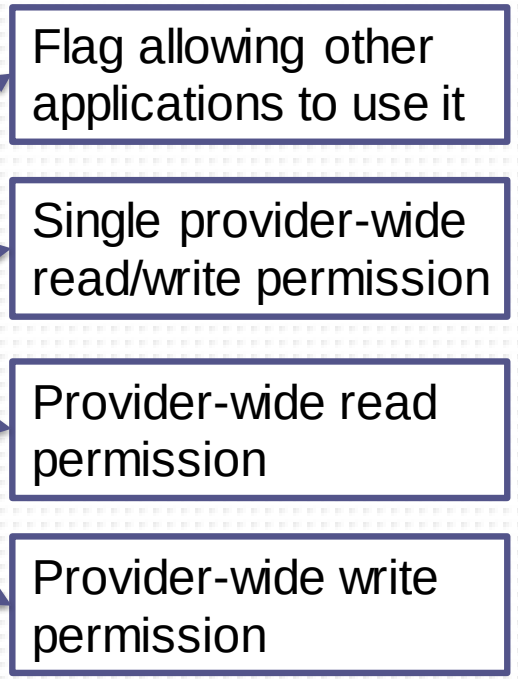
- Content providers provide a way to share data with other apps
- They encapsulate the data, and provide mechanisms for defining data security
- Use content providers if you want to expose your application data to other applications or widgets



Defining Content Provider

- ContentProvider must be defined in the manifest file using the <provider> element

```
<application>
    <provider android:authorities="list"
          android:name="string"
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:permission="string"
          android:readPermission="string"
          android:writePermission="string" >
    . . .
    </provider>
    . . .
<application>
```



Defining Content Provider cont.

- Use <permission> to declare a custom permission to be used by your provider

```
<permission android:name="com.myapp.WRITE_PERMISSION"
    android:protectionLevel="dangerous"/>
…
<application>
    <provider android:name=".MyProvider"
        android:enabled="true"
        android:exported="true"
        android:writePermission="com.myapp.WRITE_PERMISSION"
        …
    </provider>
    …
<application>
…
</manifest>
```

Requesting Content Provider Permission

- To get the permissions needed to access a provider, <uses-permission> element should be used in the manifest file
- User must approve all of the permissions the application requests

```
    …
    <uses-permission android:name="android.permission.READ_CONTACTS"  />
    …
</manifest>
```

Permissions

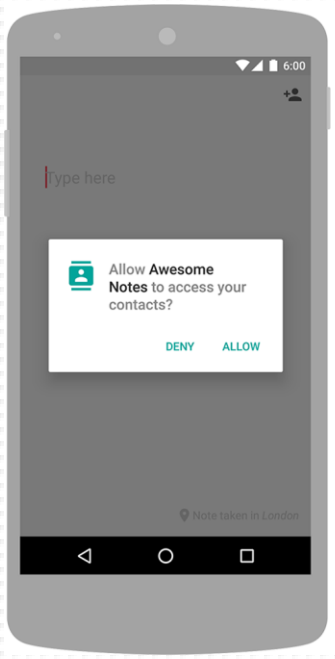
- Android apps must request permission to access
  - Sensitive user data (such as contacts and SMS)
  - Certain system features (such as camera and internet)
- Only dangerous permissions require user agreement
- App must publicize the permissions it requires by including <uses-permission> tags in the app manifest

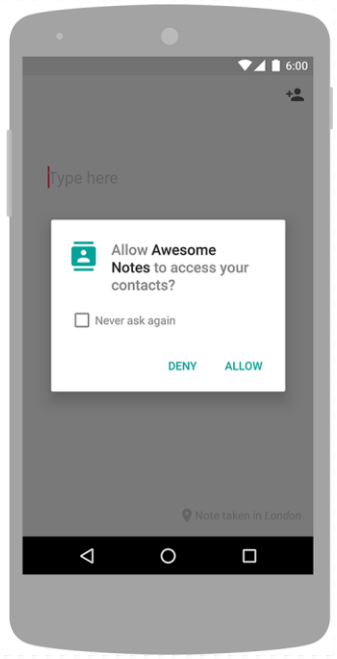
```
…
<uses-permission android:name="com.android.providers.tv.permission.READ_EPG_DATA"/>
…
</manifest>
```

- Check below link for all normal and dangerous permissions
  - https://developer.android.com/guide/topics/permissions/overview

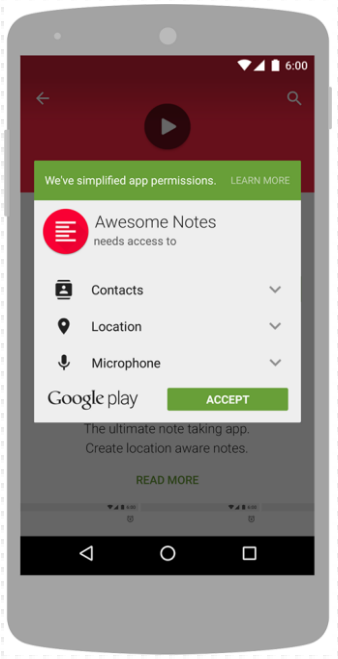
Permission Approval

- Android has 2 options to asks the user to grant permissions
  - Runtime requests (Android 6.0 and higher)
  - Install-time requests (Android 5.1.1 and below)





Runtime (initial and secondary) requests



Install-time request

Permissions for optional features

- If you request a specific hardware, your app cannot be installed to a device without related hardware
- If this hardware is optional for your app, declare this by using the <uses-feature> tag in your manifest
- If you declare android:required="false", Google Play allows your app to be installed

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

- You must call *PackageManager.hasSystemFeature()* at runtime, and gracefully disable that feature if it's not available

Android Storage

- Android provides several options to save app data
  - Internal File Storage
  - External File Storage
  - Shared Preferences
  - Databases (SQLite)

Internal File Storage

- By default, files saved to the internal storage are private to app
- When the user uninstalls app, the files are removed
  - Don’t use it if user expects files to persist independently of app

```
String filename = "myfile";
String fileContents = "Hello world!";
FileOutputStream outputStream;

try {
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
    outputStream.write(fileContents.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

External File Storage

- External storage is not a guaranteed to be accessible
- You can save two different types of files
  - Public Files (freely available to other apps)
  - Private Files (will be removed when app is uninstalled)
- Files saved to the external storage are world-readable. Even the private files can be read (e.g. when the device is connected to PC)
- To write to the public external storage, the WRITE_EXTERNAL_STORAGE permission must be requested
- If you want to just read file from external storage, the READ_EXTERNAL_STORAGE permission must be requested

External File Storage cont.

```
…
// Get the directory for the app's private pictures directory.
File file = new File(Environment.getExternalFilesDir(
        Environment.DIRECTORY_PICTURES),  albumName);
if (!file.mkdirs()) {
    Log.e(LOG_TAG, "Directory not created");
}
…
```

```
…
// Get the directory for the user's public pictures directory.
File file = new File(Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES),  albumName);
if (!file.mkdirs()) {
    Log.e(LOG_TAG, "Directory not created");
}
…
```

Internal vs External File Storage

Internal File Storage	External File Storage
It's always available	It's not always available★
Files saved here are accessible by only your app	It's world-readable; the files may be read outside of your control
When the user uninstalls the app, the system removes all the app's files	When the user uninstalls the app, the system removes **only** the private files★★

★ Many devices divide the permanent storage space into separate "internal" and "external" partitions. So even without a removable storage medium, these two storage spaces always exist, and the API behavior is the same

★★ Private files are saved via getExternalFilesDir() API

Shared Preferences

- **SharedPreferences** APIs allow you to read and write persistent key-value pairs of primitive data types
  - booleans, floats, ints, longs, and strings
- The key-value pairs are written to XML files that persist across user sessions, even if your app is killed
- You can manually specify a name for the file or use per-activity files to save your data

Shared Preferences cont.

Write to shared preferences

```
int mode = Context.MODE_PRIVATE;
SharedPreferences sharedPref = getActivity().getPreferences(mode);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt("high_score", newHighScore);
editor.commit();
```

Read from shared preferences

```
int mode = Context.MODE_PRIVATE;
SharedPreferences sharedPref = getActivity().getPreferences(mode);
int defaultValue = 0;
int highScore = sharedPref.getInt("high_score", defaultValue);
```

Databases

- Android provides full support for SQLite databases
- It is used to store structured data in a private database
- Using SQLite APIs directly is not recommended by Google
  - the SQLite APIs are fairly low-level and require a great deal of time and effort to use
- The Room persistence library provides an abstraction layer over SQLite
- Find more information below:
  - https://developer.android.com/training/data-storage/room/

QUESTIONS?