Android

Playing Audio with Media Player

Playing Sound Effects

- Find sound files such as .WAV, .MP3
- put sound files in project folder **app/src/main/res/raw**
- in Java code, refer to audio file as R.raw.***filename***
  - (don't include the extension; R.raw.foo for foo.mp3)
  - use simple file names with only letters and numbers

- Load and play clips using Android's MediaPlayer class

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.filename);
mp.start();
```

  - other methods: **stop**, **pause**, isLooping, **isPlaying**, getCurrentPosition, **release**, seekTo, setDataSource, setLooping

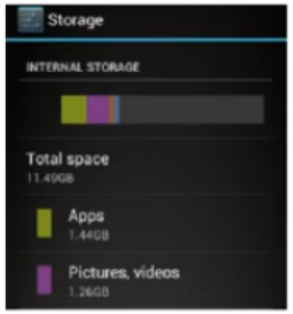
Data Files and Storage

Dictionary

Files and Storage

- Android can read/write files from two locations:
  - **internal** and **external** storage.
  - Both are **persistent** storage; data remains after power-off / reboot.

- **internal storage**: Built into the device.
  - guaranteed to be present
  - typically smaller (~1-4 gb)
  - can't be expanded or removed
  - specific and private to each app
  - wiped out when the app is uninstalled



File and Streams

- java.io.**File** - Objects that represent a file or directory.
  - methods: canRead, canWrite, create, delete, exists, getName, getParent, getPath, isFile, isDirectory, lastModified, length, listFiles, mkdir, mkdirs, renameTo

- java.io.Input**Stream**, OutputStream - Stream objects represent flows of data bytes from/to a source or destination.
  - Could come from a file, network, database, memory, ...
  - Normally not directly used; they only include low-level methods for reading/writing a byte (character) at a time from the input.
  - Instead, a stream is often passed as parameter to other objects like java.util.**Scanner**, java.io.**BufferedReader**, java.io.**PrintStream** to do the actual reading / writing.

Using Internal Storage

- An activity has methods you can call to read/write files:
  - `getFilesDir()` - returns internal directory for your app
  - `getCacheDir()` - returns a "temp" directory for scrap files
  - `getResources().openRawResource(R.raw.id)` - read an input file from `res/raw/`
  - `openFileInput("name", mode)` - opens a file for reading
  - `openFileOutput("name", mode)` - opens a file for writing

- You can use these to read/write files on the device.
  - many methods return standard java.io.**File** objects
  - some return java.io.**InputStream** or **OutputStream** objects, which can be used with standard classes like Scanner, BufferedReader, and PrintStream to read/write files (see Java API)

Internal Storage Example-1

```
// read a file, and put its contents into a TextView
// (assumes hello.txt file exists in res/raw/ directory)
Scanner scan = new Scanner(
        getResources().openRawResource(R.raw.hello));
String allText = "";   // read entire file
while (scan.hasNextLine()) {
    String line = scan.nextLine();
    allText += line;
}
myTextView.setText(allText);
scan.close();
```

Internal Storage Example-2

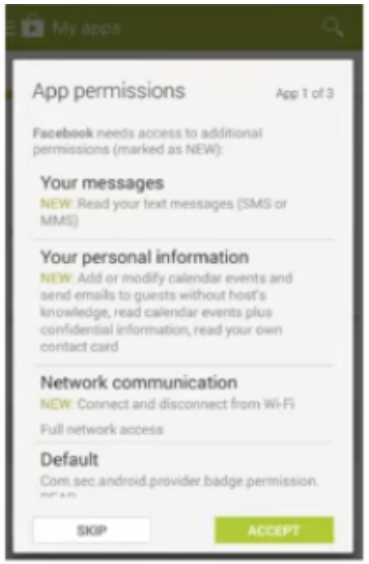
```
// write a short text file to the internal storage
PrintStream output = new PrintStream
        openFileOutput("out.txt", MODE_PRIVATE));
output.println("Hello, world!");
output.println("How are you?");
output.close();
...
// read the same file, and put its contents into a TextView
Scanner scan = new Scanner(
        openFileInput("out.txt", MODE_PRIVATE));
String allText = "";   // read entire file
while (scan.hasNextLine()) {
    String line = scan.nextLine();
    allText += line;
}
myTextView.setText(allText);
scan.close();
```

External Stroge

- **external storage**: Card that is inserted into the device.
  (such as a MicroSD card)
  - can be much larger than internal storage (~8-32 gb)
  - can be removed or transferred to another device if needed
  - may not be present, depending on the device
  - read/writable by other apps and users; not private to your app
  - *not* wiped when the app is uninstalled, except in certain cases

External Storage Permission

- If your app needs to read/write the device's external storage, you must explicitly request **permission** to do so in your app's **AndroidManifest.xml** file.
  - On install, the user will be prompted to confirm your app permissions.




```
<manifest ...>
    <uses-permission
     android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
     android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

Using External Storage

- Methods to read/write external storage:
  - `getExternalFilesDir("name")` -
    returns "private" external directory for your app with the given name
  - `Environment.getExternalStoragePublicDirectory(name)` -
    returns public directory for common files like photos, music, etc.
    - pass constants for ***name*** such as Environment.DIRECTORY_ALARMS, DIRECTORY_DCIM, DIRECTORY_DOWNLOADS, DIRECTORY_MOVIES, DIRECTORY_MUSIC, DIRECTORY_NOTIFICATIONS, DIRECTORY_PICTURES, DIRECTORY_PODCASTS, DIRECTORY_RINGTONES

- You can use these to read/write files on the external storage.
  - the above methods return standard java.io.**File** objects
  - these can be used with standard classes like Scanner, BufferedReader, and PrintStream to read/write files (see Java API)

Checking External Storage is Available??

```
/* Checks if external storage is available
 * for reading and writing */
public boolean isExternalStorageWritable() {
    return Environment.MEDIA_MOUNTED.equals(
            Environment.getExternalStorageState());
}

/* Checks if external storage is available
 * for reading */
public boolean isExternalStorageReadable() {
    return isExternalStorageWritable() ||
            Environment.MEDIA_MOUNTED_READ_ONLY.equals(
                Environment.getExternalStorageState());
}
```

Accessing Web Data

- To read data from the web, first request the INTERNET permission in your **AndroidManifest.xml**:

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

- Then you can use the standard `java.net.URL` class to connect to a file or page at a given URL and read its data:

```
URL url = new URL("http://foobar.com/example.txt");
Scanner scan = new Scanner(url.openStream());
while (scan.hasNextLine()) {
    String line = scan.nextLine();
    ...
}
```

Text to Speech

- **text-to-speech**: Allows Android device to speak an audible message based on a text string.
  - Not installed by default on some devices. To install, click Settings → Language and Input → Text to speech output → Google text-to-speech engine "settings" icon → Install voice data → Languages

- In general, text-to-speech on Android is simple:
  - create a `TextToSpeech` object
  - call its `speak` method

- But there are a few details that require us to discuss some advanced features of Java.

Text to Speech Class

- TextToSpeech constructor and methods:
  - `new TextToSpeech(activity, listener)` - constructor
  - `getVoice, getVoices, setVoice` - change speaking voice
  - `getLanguage, setLanguage` - sets language used
  - `getPitch, setPitch` - sets vocal tone used
  - `isSpeaking` - returns true if speaking
  - `shutdown` - kills TTS engine
  - `speak(text, mode, params)` - speaks given text aloud
  - `stop` - halts any speech
  - `synthesizeToFile(text, params, filename)` - speaks to file

The Speak Method

- The speak method accepts three parameters *:
  - the text to speak aloud, as a String.
  - the mode to use for speaking, one of:

    `TextToSpeech.QUEUE_ADD` : Speak after any other text is done.

    `TextToSpeech.QUEUE_FLUSH` : Stop any other text and speak immediately.
  - a Map of parameters (we don't need any, so we can pass `null`).

    ** (Android 5 introduces a different version of speak(), but we'll ignore it.)*

```
// speak text aloud, if my init boolean flag is set
if (myTTSisReady) {
    tts.speak("Hello, world!",
              TextToSpeech.QUEUE_FLUSH, null);
}
```

Using the Camera

Photography

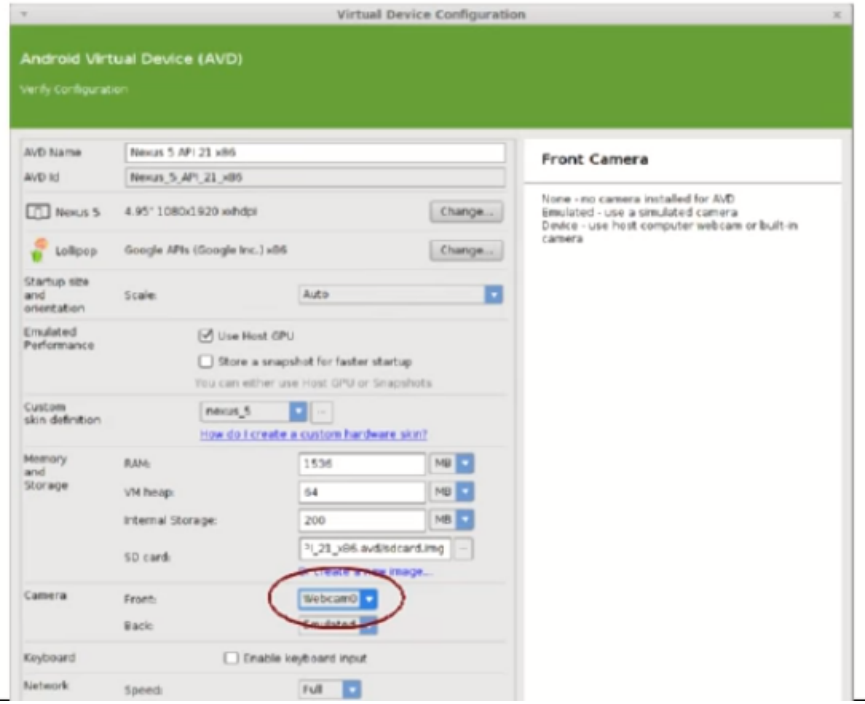
Requesting Camera Permission

- If your app uses the device camera, you must explicitly state that it will use that hardware in your app's **AndroidManifest.xml** file.

```
<manifest ...>
    <uses-permission
        android:name="android.permission.CAMERA" />
    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />
    ...
</manifest>
```

AVD and Camera

- It is difficult to emulate a camera in the Android Virtual Device.
- In AVD Manager, you can try setting the device to use your laptop's webcam as the virtual camera, but it may crash.
  - AVD Manager → Edit → Advanced Settings → Camera
- Alternatively, just deploy your app to a physical device to test camera functionality.



Camera Code Example

- Example that takes a picture and places it into an ImageView:

```
private static final int REQ_CODE_TAKE_PICTURE = 90210;
...
Intent picIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(picIntent, REQ_CODE_TAKE_PICTURE);
...

@Override
protected void onActivityResult(int requestCode, int resultCode,
                                Intent intent) {
    if (requestCode == REQ_CODE_TAKE_PICTURE
            && resultCode == RESULT_OK) {
        Bitmap bmp = (Bitmap) intent.getExtras().get("data");
        ImageView img = (ImageView) findViewById(R.id.camera_image);
        img.setImageBitmap(bmp);
    }
}
```

Thanks for your attention!