Android Developer Fundamentals V2

# Preferences and settings

# Files

# Android File System

- External storage -- Public directories
- Internal storage -- Private directories for just your app

Apps can browse the directory structure

Structure and operations similar to Linux and java.io

# Internal storage

- Always available
- Uses device's filesystem
- Only your app can access files, unless explicitly set to be readable or writable
- On app uninstall, system removes all app's files from internal storage

# External storage

- Not always available, can be removed
- Uses device's file system or physically external storage like SD card
- World-readable, so any app can read
- On uninstall, system does not remove files private to app

# When to use internal/external storage

**Internal is best when**

- you want to be sure that neither the user nor other apps can access your files

**External is best for files that**

- don't require access restrictions and for
- you want to share with other apps
- you allow the user to access with a computer

# Save user's file in shared storage

- Save new files that the user acquires through your app to a public directory where other apps can access them and the user can easily copy them from the device
- Save external files in public directories

# Internal Storage

# Internal Storage

- Uses private directories just for your app
- App always has permission to read/write
- Permanent storage directory—getFilesDir()
- Temporary storage directory—getCacheDir()

# Creating a file

```
File file = new File(
    context.getFilesDir(), filename);
```

Use standard java.io file operators or streams to interact with files

# 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`***`" )` - 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)

# Example 1

```
// read a file, and put its contents into a TextView
// (assumes hello.txt file exists in res/raw/ directory)
InputStream ins = getResources().openRawResource(getResources().
getIdentifier("FILENAME_WITHOUT_EXTENSION", "raw", getPackageName()));

Scanner scan = new Scanner(ins);
String allText = ""; // read entire file
while (scan.hasNextLine()) {
        String line = scan.nextLine();
        allText += line;
}
myTextView.setText(allText);
scan.close();
```

--Just we can read the file only from the raw folder

# 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"));
String allText = ""; // read entire file
while (scan.hasNextLine()) {
        String line = scan.nextLine();
        allText += line;
}
myTextView.setText(allText);
scan.close();
```

# External Storage

# External Storage

- On device or SD card
- Set permissions in Android Manifest
  - Write permission includes read permission


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

# Always check availability of storage

```
/* 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());
}
```

# Example external public directories

- DIRECTORY_ALARMS and DIRECTORY_RINGTONES
  For audio files to use as alarms and ringtones
- DIRECTORY_DOCUMENTS
  For documents that have been created by the user
- DIRECTORY_DOWNLOADS
  For files that have been downloaded by the user
- ..... developer.android.com/reference/android/os/Environment.html

# Accessing public external directories

1. Get a path getExternalStoragePublicDirectory()
2. Create file

```
File path = Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES);

File file = new File(path, "DemoPicture.jpg");
```

```
String path = Environment.getExternalStoragePublicDirectory(
   Environment.DIRECTORY_DOCUMENTS) + File.separator + "myFolder";
File outputDir = new File(path);
if(!outputDir.exists()) outputDir.mkdir();
File f = new File(outputDir + "/" + this.getString(R.string.fileName));
```

# How much storage left?

- If there is not enough space, throws IOException
- If you know the size of the file, check against space
  - getFreeSpace()
  - getTotalSpace().
- If you do  not know how much space is needed
  - try/catch IOException

# Delete files no longer needed

- External storage

```
myFile.delete();
```

- Internal storage

```
myContext.deleteFile(fileName);
```

# Do not delete the user's files!

When the user uninstalls your app, your app's private storage directory and all its contents are deleted

***Do not use private storage for content that belongs to the user!***

For example

- Photos captured or edited with your app
- Music the user has purchased with your app

# Private external directory

Methods to read/write external storage:

`getExternalFilesDir("`***`name`***`")` - returns "private" external directory for your app with the given name

– 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)

```
// write short data to app-specific external storage
File outDir = getExternalFilesDir(null);   // root dir
File outFile = new File(outDir, "example.txt");
PrintStream output = new PrintStream(outFile);
output.println("Hello, world!");
output.close();
}
```

# Example

```
// read list of pictures in external storage
File picsDir = Environment.getExternalStoragePublicDirectory(
Environment.DIRECTORY_DCIM);
File dirp = new File(picsDir + File.separator + "CAMERA");
if (dirp.exists()) {
            String s = "";
            File[ ] fs = dirp.listFiles();
          for (File file : fs)
                  s += file.getName() + "\n";
            edt.setText(s);
}
else
            edt.setText("Folder does not exist")
```

# 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();
    ...
}
```

# END