File I/O

Upon completion of this module, a student will be able to

- understand and explain the difference between internal storage, external storage, and cache
- get a File object from the three different storage locations
- write to a file
- read from a file
- store and access objects using serialization

Assignment

- Task
    - Take the threading assignment from S03M01 and pull the encrypted contents from a file and then store it in another file once it has been decrypted.
- Repo
    - https://github.com/LambdaSchool/Android_AsynTasks_FileIO
- Submission
    - Copy your new project into a fork of this repo and submit a pull request
- Challenge
    - Improve the app with the suggestions in the readme

A Student Can

understand and explain the difference between internal storage, external storage, and cache

Internal and External Storage

- Always available.
- Private Files
- Removed on Uninstall

- Not consistently available
- Public Files
- Only removed if getExternalFilesDir() is used

Permissions

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

- Public File System
- Older Devices (< API 19)
- Permission must be Granted

A Student Can

get a File object

Cache

- Temporary file storage
- Cleared whenever the system needs it

```
private File getTempFile(Context context, String url) {
    File file;
    try {
        String fileName = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(fileName, null, context.getCacheDir());
    } catch (IOException e) {
        // Error while creating file
    }
    return file;
}
```

Getting a File

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

// Cache File Storage
// use existing file
File file = new File(context.getCacheDir(), filename);
// create new file
File file = File.createTempFile(fileName, null, context.getCacheDir());

//External File Storage
File directory = new File(Environment.getExternalStoragePublicDirectory(
          Environment.DIRECTORY_PICTURES), albumName);
if (!directory.mkdirs()) {
    Log.e(LOG_TAG, "Directory not created");
}
File file = new File(directory, filename);
// to get the private external directory
context.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
```

- Internal
  - getFilesDir
- Cache
  - getCacheDir
- External
  - getExternalFilesDir
  - getExternalStoragePublicDirectory

Check External Storage

- Check State
  - Mounted Writable
  - Mounted Read Only

```java
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

A Student Can

write to a file

Working with Text

- FileWriter
  - Construct Writer
  - Write String(s)
  - Close

```
FileWriter writer;
try {
    writer = new FileWriter(file);
    writer.write(text);
    writer.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Working with Bytes

```
FileOutputStream   fileOutputStream;
try {
    fileOutputStream = new FileOutputStream(file.getPath());
    fileOutputStream.write(bytes);
    fileOutputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

- FileOutputStream
  - Construct Stream
    - Use File path
  - Write bytes
  - Close

A Student Can

read from a file

Working with Text

- FileReader
  - Construct Reader
  - Read String
  - Close

```
FileReader    reader;
StringBuilder readData = new StringBuilder();
try {
    reader = new FileReader(file);
    int next = reader.read();
    while (next != -1) {
        readData.append((char) next);
        next = reader.read();
    }
    reader.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Working with Bytes

```
FileInputStream fileInputStream;
ArrayList<Byte> readData = new ArrayList<>();
try {
    fileInputStream = new FileInputStream(file.getPath());
    do {
        readData.add((byte)fileInputStream.read());
    } while (readData.get(readData.size() - 1) != -1);
    readData.remove(readData.size() - 1);
    fileInputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

- FileInputStream
  - Construct Stream
    - Use File path
  - Read bytes
    - Add to list in while loop
  - Close

A Student Can

store and access objects using serialization

Object Serialization

- implements Serializable
- Some classes are not serializable
  - Must convert them to something else first

```
import java.io.Serializable;

public class MyClass implements Serializable {
    ...
}
```

Working with Serialized Objects

```
// store the object
FileOutputStream   fileOutputStream;
ObjectOutputStream objectOutputStream;
MyClass            object;
try {
    fileOutputStream = new FileOutputStream(file.getPath());
    objectOutputStream = new ObjectOutputStream(fileOutputStream);
    objectOutputStream.writeObject(object);
    objectOutputStream.close();
    fileOutputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}

// retreive the object
FileInputStream   fileInputStream;
ObjectInputStream objectInputStream;
MyClass           storedObject;
try {
    fileInputStream = new FileInputStream(file.getPath());
    objectInputStream = new ObjectInputStream(fileInputStream);
    storedObject = (MyClass)objectInputStream.readObject());
    objectInputStream.close();
    fileInputStream.close();
} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

- ObjectOutputStream
- ObjectInputStream