5COSC005W MOBILE APPLICATION DEVELOPMENT
Lecture 8: Data Storage - Part II

Dr Dimitris C. Dracopoulos

Module Web page:

https://dracopd.users.ecs.westminster.ac.uk/DOCUM/courses/5cosc005w/5cosc005w.html

Data Storage

Storage Options

Storing data

- SQLite Databases—Structured data in a private database
- Shared Preferences—Private primitive data in key-value pairs
- Internal Storage—Private data on device memory
- External Storage—Public data on device or external storage
- Content Providers—Store privately and make available publicly

Storing data beyond Android

- Network Connection—On the web with your own server
- Cloud Backup—Back up app and user data in the cloud
- Firebase Realtime Database—Store and sync data with NoSQL cloud database across clients in realtime

SQLite Database

- Ideal for repeating or structured data, such as contacts
- Android provides SQL-like database
- Covered in previous week

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

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

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

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");
```

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

Shared Preferences

Shared Preferences

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage

- Covered in later here

Other Storage Options

Use Firebase to store and share data

Store and sync data with the Firebase cloud database

Data is synced across all clients, and remains available when your app goes offline

firebase.google.com/docs/database/

Database

Store and sync data in realtime across all connected clients

Firebase Realtime Database

- Connected apps share data
- Hosted in the cloud
- Data is stored as JSON
- Data is synchronized in realtime to every connected client

Network Connection

- You can use the network (when it's available) to store and retrieve data on your own web-based services

- Use classes in the following packages
  - java.net.*
  - android.net.*

Shared Preferences

What is Shared Preferences?

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- SharedPreference class provides APIs for reading, writing, and managing this data
- Save data in onPause()
  restore in onCreate()

Shared Preferences AND Saved Instance State

- Small number of key/value pairs
- Data is private to the application

Shared Preferences vs. Saved Instance State

- Persist data across user sessions, even if app is killed and restarted, or device is rebooted
- Data that should be remembered across sessions, such as a user's preferred settings or their game score
- Common use is to store user preferences

- Preserves state data across activity instances in same user session
- Data that should not be remembered across sessions, such as the currently selected tab or current state of activity.
- Common use is to recreate state after the device has been rotated

Creating Shared Preferences

- Need only one Shared Preferences file per app
- Name it with package name of your app—unique and easy to associate with app
- MODE argument for getSharedPreferences() is for backwards compatibility—use only MODE_PRIVATE to be secure

getSharedPreferences()

```
private String sharedPrefFile =
    "com.example.android.hellosharedprefs";

mPreferences =
    getSharedPreferences(sharedPrefFile,
                         MODE_PRIVATE);
```

Saving Shared Preferences

- SharedPreferences.Editor interface
- Takes care of all file operations
- put methods overwrite if key exists
- apply() saves asynchronously and safely

SharedPreferences.Editor

```
@Override
protected void onPause() {
   super.onPause();
   SharedPreferences.Editor preferencesEditor =
       mPreferences.edit();
   preferencesEditor.putInt("count", mCount);
   preferencesEditor.putInt("color", mCurrentColor);
   preferencesEditor.apply();
}
```

Restoring Shared Preferences

- Restore in onCreate() in Activity
- Get methods take two arguments—the key, and the default value if the key cannot be found
- Use default argument so you do not have to test whether the preference exists in the file

Getting data in onCreate()

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

mCount = mPreferences.getInt("count", 1);
mShowCount.setText(String.format("%s", mCount));

mCurrentColor = mPreferences.getInt("color", mCurrentColor);
mShowCount.setBackgroundColor(mCurrentColor);

mNewText = mPreferences.getString("text", "");
// ...
```

Clearing

- Call clear() on the SharedPreferences.Editor and apply changes

- You can combine calls to put and clear. However, when you apply(), clear() is always done first, regardless of order!

clear()

```
SharedPreferences.Editor preferencesEditor =
                    mPreferences.edit();
preferencesEditor.clear();
preferencesEditor.apply();
```

Listing to Changes

Listening to changes

- Implement interface SharedPreference.OnSharedPreferenceChangeListener
- Register listener with registerOnSharedPreferenceChangeListener()
- Register and unregister listener in onResume() and onPause()
- Implement on onSharedPreferenceChanged() callback

Interface and callback

```
public class SettingsActivity extends AppCompatActivity
    implements OnSharedPreferenceChangeListener { ...

    public void onSharedPreferenceChanged(
        SharedPreferences sharedPreferences, String key) {
        if (key.equals(MY_KEY)) {
            // Do something
        }
    }
}
```

Creating and registering listener

```
SharedPreferences.OnSharedPreferenceChangeListener listener =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
  public void onSharedPreferenceChanged(
     SharedPreferences prefs, String key) {
        // Implement listener here
  }
};
prefs.registerOnSharedPreferenceChangeListener(listener);
```

You need a STRONG reference to the listener

- When registering the listener the preference manager does not store a strong reference to the listener
- You must store a strong reference to the listener, or it will be susceptible to garbage collection
- Keep a reference to the listener in the instance data of an object that will exist as long as you need the listener