Advanced Mobile Application Development
Week 13: Fragments and Persistent Data

There are multiple data persistence approaches on Android. The approach you pick should be based on

- What kind of data you need to store
- How much space your data requires
- Whether the data should be private to your app or accessible to other apps and the user

Device Storage

All Android devices have two file storage areas: "internal" and "external" storage. These names come from the early days of Android, when most devices offered built-in non-volatile memory (internal storage), plus a removable storage medium such as a micro SD card (external storage). Many devices now divide the permanent storage space into separate "internal" and "external" partitions.
(covered in chapter 17)

Internal file storage

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

Pros

- Always available
- By default files saved are private to your app
- Other apps and the user can't access the files
- Files are removed when the user uninstalls your app

Cons

- Hard to share data
- Internal storage might have limited capacity

For temporary storage you should use an internal cache file.

External file storage

External storage is best for files that don't require access permissions or that you want to share with other apps or users. It's called external because it's not a guaranteed to be accessible—it is a storage space that users can mount to a computer as an external storage device, and it might even be physically removable (such as an SD card). Most often, you should use external storage for user data that should be accessible to other apps and saved even if the user uninstalls your app, such as captured photos or downloaded files.

- Similar to internal storage but the data files are world-readable and can be modified by the user without developer or user permission when they enable USB mass storage to transfer files on a computer.
- Only available when the storage is accessible
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from getExternalFilesDir().

After you request storage permissions and verify that storage is available, you can save two different types of files:

- Public files: Files that should be freely available to other apps and to the user. When the user uninstalls your app, these files should remain available to the user.
- Private files: Files that rightfully belong to your app and will be deleted when the user uninstalls your app. Although these files are technically accessible by the user and other apps because they are on the external storage, they don't provide value to the user outside of your app.

Database
Android provides full support for SQLite databases. Any database you create is accessible only by your app.

- A SQL database is a good choice for a large amount of structured data
- Any database you create is accessible only by your app
- Covered in chapter 18
- Android also includes the Room library which provides an abstraction layer over SQLLite https://developer.android.com/training/data-storage/room/index.html
    - Highly recommended over SQLLite
    - Still requires SQL
    - Need to add the components to your gradle file

Shared Preferences (different than user preferences)
https://developer.android.com/training/data-storage/shared-preferences.html
SharedPreferences is a good choice if you don't need to store a lot of data and it doesn't require structure.

- Read and write key-value pairs of primitive data types: boolean, float, int, long, string, and stringset.
- Persistent across user sessions even if your app is killed
- Use **getPreferences()** if you're only using one shared preference file in an activity as this uses the activity name as the preference set name
- Use **getSharedPreferences()** if you need multiple shared preferences files each with a unique set name
- Usually **MODE_PRIVATE** as the security visibility parameter so the preference is private and no other application can access it.
- Covered in the beginning of chapter 19

Writing to a shared preferences file

1. create a **SharedPreferences.Editor** by calling **edit()** on your SharedPreferences object
2. Add the key-value pairs using methods like **putInt(), putString(),** and **putStringSet()**
3. Call **commit()** to save the changes

Reading from a shared preferences file

1. create a **SharedPreferences.Editor** by calling **edit()** on your SharedPreferences object
2. Read in the key-value pairs using methods like **getInt(), getString(),** and **getStringSet()**

Except for some types of files on external storage, all these options are intended for app-private data—the data is not naturally accessible to other apps. If you want to share files with other apps, you should use the FileProvider API.
If you want to expose your app's data to other apps, you can use a ContentProvider. Content providers give you full control of what read/write access is available to other apps, regardless of the storage medium you've chosen for the data (though it's usually a database).

Superheroes
We're going to add data persistence to our superheroes app using shared preferences.
Update Hero.java to store our data.

```
public void storeHeroes(Context context, long universeId){
    //get access to a shared preferences file
    SharedPreferences sharedPrefs = context.getSharedPreferences("Superheroes",
Context.MODE_PRIVATE);
    //create an editor to write to the shared preferences file
    SharedPreferences.Editor editor = sharedPrefs.edit();
    //create a set
    Set<String> set = new HashSet<String>();
    //add heroes to the set
    set.addAll(heroes.get((int) universeId).getSuperheroes());
    //pass the key/value pair to the shared preference file
    editor.putStringSet(heroes.get((int) universeId).getUniverse(), set);
    //save changes
    editor.commit();
}

import android.content.SharedPreferences;
import android.content.Context;
import java.util.HashSet;
import java.util.Set;
```

In HeroDetailFragment.java we need to save our hero list whenever we add or delete a hero.
Update addhero() and onContextItemSelected() after you call notifyDataSetChanged()

```
Hero.heroes.get((int) universeId).storeHeroes(getActivity(), universeId);
```

To test, exit out of the app and restart it. Note that the list order might be different.
(DC: Black Canary; Marvel: Jessica Jones)

Now to read in from shared preferences.
Update MainActivity.java to load the data.

Update onCreateView()

```
//load data
if(Hero.heroes.isEmpty()) {
    loadHeroes(this);
}
```

Load the data from shared preferences if there's data saved, otherwise we load our XML file.

```
public void loadHeroes(Context context) {
    //get access to a shared preferences file
    SharedPreferences sharedPrefs = context.getSharedPreferences("Superheroes",
Context.MODE_PRIVATE);
    //create an editor to read from the shared preferences file
    SharedPreferences.Editor editor = sharedPrefs.edit();
```

```
    //load all data into a Map
    Map<String, ?> alldata = sharedPrefs.getAll();
    //if there's data in the map
    if (alldata.size() > 0) {
        for (Map.Entry<String, ?> entry : alldata.entrySet()) {
            //key is the universe name
            String newuniverse = (String) entry.getKey();
            ArrayList<String> herolist = new ArrayList<String>();
            //add the heroes String set to the ArrayList
            herolist.addAll(sharedPrefs.getStringSet(newuniverse, null));
            //create new Hero object
            Hero new_hero = new Hero(newuniverse, herolist);
            //add hero
            Hero.heroes.add(new_hero);
        }
    } else{ //no data, load XML file
        try {
            loadXML(this);
            } catch (XmlPullParserException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```