13. Lesson 18/04/23

Shared Preferences

What are Shared Preferences?

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

- Preference file is accessible to all the components of your app, but it is not accessible to other apps.
- SharedPreference provides APIs for reading, writing, and managing this data.
- Save data in *OnPause()*
- Restore sata in *OnCreate()*

Shared Preferences AND Saved Instance State

For both:

- Data represented by a samll number of **key/value pairs.**
- **Data is private** to the application.

Shared Preferences vs. Saved Instance State

Shared Preferences:

- **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.**
- Common use is to **store preferences.**

Saved Instance State:

- **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 the app).

getSharedPreferences()

```
private String sharedPrefFile = "com.example.android.hellosharedprefs";
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

MODE argument: for getSharedPreferences() is for **backwards compatibility.** Use only MODE_PRIVATE to be secure.

Saving Shared Preferences

- ***SharedPreferences.Editor*** interface —> It takes care of all file operations.
- Put methods overwrite if the key exists.
- *apply()* saver asynchronously and safely.

Save preferences in the *onPause()* state of the activity lifecycle using the *SharedPreferences.Editor* interface.

Get a *SharedPreferences.Editor*

- The editor takes care of all the file operations for you.

Add key/value pairs to the editor using the “put” method appropriate for the data type:

- F.I.: *putInt()* or *putString()*
- These methods will overwrite previously existing values of an existing key.

Call *apply()* to write out your changes

- The *apply()* method saves the preferences asynchronously, off of the UI thread

You don’t need to worry about Android component lifecycles and their interaction with apply() writing to disk.

- The framework makes sure in-flight disk writes from *apply()* complete before switching states.

SharedPreferences.Editor

```
public class MainActivity extends AppCompatActivity {
  private SharedPreferences mPreferences;
  …
  @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**
  - 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()

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  String sharedPrefFile = "com.example.simplesavingsapp";
  mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
  mCount = mPreferences.getInt("count", 1);
  mCurrentColor = mPreferences.getInt("color", 0);
  … // use that values e.g., for initializing UI widgets
}
```

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();
```