9.1: Shared preferences

Contents:

Shared preferences allow you to store small amounts of primitive data as key/value pairs in a file on the device. To get a handle to a preference file, and to read, write, and manage preference data, use the `SharedPreferences` class. The Android framework manages the shared preferences file itself. The file is accessible to all the components of your app, but it is not accessible to other apps.

For managing large amounts of data, use an SQLite database or other suitable storage option, which is discussed in a later chapter.

Shared preferences vs. saved instance state

In a previous chapter you learned about preserving state using saved instance states. Here is a comparison between the two:

Shared preferences	**Saved instance state**
Persists across user sessions, even if your app is stopped and restarted, or if the device is rebooted.	Preserves state data across activity instances in the same user session.
Used for data that should be remembered across user sessions, such as a user's preferred settings or their game score.	Used for data that should not be remembered across sessions, such as the currently selected tab, or any current state of an activity.
Represented by a small number of key/value pairs.	Represented by a small number of key/value pairs.
Data is private to the app.	Data is private to the app.
Common use is to store user preferences.	Common use is to recreate state after the device has been rotated.

> **Note:** The `SharedPreference` APIs are different from the `Preference` APIs. The `Preference` APIs can be used to build a user interface for a settings page, and they use shared preferences for their underlying implementation. For more information on settings and the `Preference` APIs, see Settings.

Creating a shared preferences file

You need only one shared preferences file for your app, and it is customarily named with the package name of your app. This makes its name unique and easily associated with your app.

You create the shared preferences file in the `onCreate()` method of your main activity and store it in a member variable.

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

The mode argument is required, because older versions of Android had other modes that allowed you to create a world-readable or world-writable shared preferences file. These modes were deprecated in API 17, and are now **strongly discouraged** for security reasons. If you need to share data with other apps, use a service or a content provider.

Saving shared preferences

You save preferences in the `onPause()` state of the activity lifecycle using the `SharedPreferences.Editor` interface.

1. Get a `SharedPreferences.Editor`. The editor takes care of all the file operations for you. When two editors are modifying preferences at the same time, the last one to call `apply()` wins.
2. Add key/value pairs to the editor using the "put" method appropriate for the data type, for example, `putInt()` or `putString()`. These methods will overwrite previously existing values of an existing key.
3. Call `apply()` to write out your changes. The `apply()` method saves the preferences asynchronously, off of the UI thread. The shared preferences editor also has a `commit()` method to synchronously save the preferences. The `commit()` method is discouraged as it can block other operations. As `SharedPreferences` instances are singletons within a process, it's safe to replace any instance of `commit()` with `apply()` if you were already ignoring the return value.

   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.

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

Restoring shared preferences

You restore shared preferences in the `onCreate()` method of your activity. The "get" methods such as `getInt()` or `getString()` take two arguments—one for the key and one for the default value if the key cannot be found. Using the default argument, you don't have to test whether the preference exists in the file.

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
if (savedInstanceState != null) {
    mCount = mPreferences.getInt("count", 1);
    mShowCount.setText(String.format("%s", mCount));

    mCurrentColor = mPreferences.getInt("color", mCurrentColor);
    mShowCount.setBackgroundColor(mCurrentColor);
} else { ... }
```

Clearing shared preferences

To clear all the values in the shared preferences file, call the `clear()` method on the shared preferences editor and apply the changes.

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
preferencesEditor.putInt("number", 42);
preferencesEditor.clear();
preferencesEditor.apply();
```

You can combine calls to put and clear. However, when applying the preferences, the clear is always done first, regardless of whether you called clear before or after the put methods on this editor.

Listening for preference changes

There are several reasons you might want to be notified as soon as the user changes one of the preferences. In order to receive a callback when a change happens to any one of the preferences, implement the `SharedPreference.OnSharedPreferenceChangeListener` interface and register the listener for the `SharedPreferences` object by calling `registerOnSharedPreferenceChangeListener()`.

The interface has only one callback method, `onSharedPreferenceChanged()`, and you can implement the interface as a part of your activity.

```
public class SettingsActivity extends PreferenceActivity
                              implements OnSharedPreferenceChangeListener {
    public static final String KEY_PREF_SYNC_CONN =
       "pref_syncConnectionType";

    // ...

    public void onSharedPreferenceChanged(
                              SharedPreferences sharedPreferences,
                              String key) {
        if (key.equals(KEY_PREF_SYNC_CONN)) {
            Preference connectionPref = findPreference(key);
            // Set summary to be the user-description for
            // the selected value
            connectionPref.setSummary(
               sharedPreferences.getString(key, ""));
        }
    }
}
```

In this example, the method checks whether the changed setting is for a known preference key. It calls `findPreference()` to get the `Preference` object that was changed so it can modify the item's summary to be a description of the user's selection.

For proper lifecycle management in the activity, register and unregister your `SharedPreferences.OnSharedPreferenceChangeListener` during the `onResume()` and `onPause()` callbacks, respectively:

```
@Override
protected void onResume() {
    super.onResume();
    getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
}

@Override
protected void onPause() {
    super.onPause();
    getPreferenceScreen().getSharedPreferences()
            .unregisterOnSharedPreferenceChangeListener(this);
}
```

Hold a reference to the listener

When you call `registerOnSharedPreferenceChangeListener()`, the preference manager does not currently store a reference to the listener. You must hold onto a reference to the listener, or it will be susceptible to garbage collection. Keep a reference to the listener as a class member variable in an object such as an activity that will exist as long as you need the listener.

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

prefs.registerOnSharedPreferenceChangeListener(listener);
```