Lesson 9.1: Shared preferences

Introduction

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.

The data you save to shared preferences is different from the data in the saved activity state, which you learned about in an earlier chapter:

- Data in a saved activity instance state is retained across activity instances in the same user session.
- Shared preferences persist across user sessions. Shared preferences persist even if your app stops and restarts, or if the device reboots.

Use shared preferences only when you need to save a small amount data as simple key/value pairs. To manage larger amounts of persistent app data, use a storage method such as the Room library or an SQL database.

What you should already know

You should be familiar with:

- Creating, building, and running apps in Android Studio.
- Designing layouts with buttons and text views.
- Using styles and themes.
- Saving and restoring activity instance state.

What you'll learn

You will learn how to:

- Identify what shared preferences are.
- Create a shared preferences file for your app.
- Save data to shared preferences, and read those preferences back again.

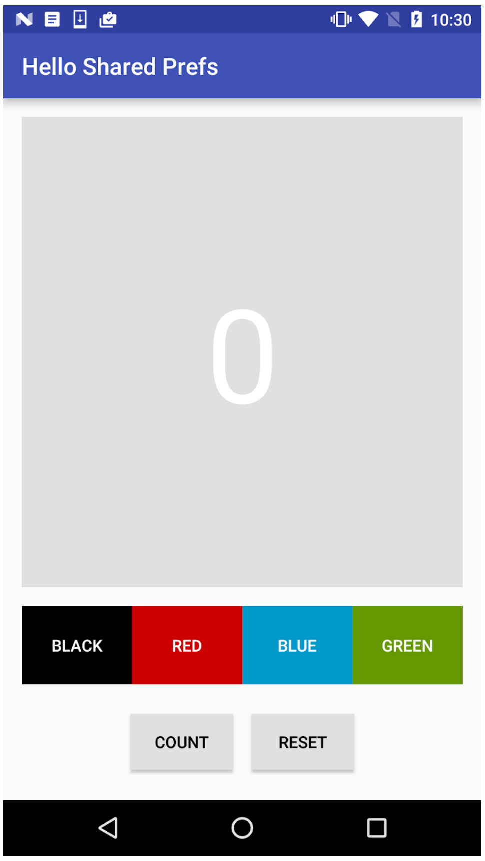
- Clear the data in the shared preferences.

What you'll do

- Update an app so it can save, retrieve, and reset shared preferences.

App overview

The HelloSharedPrefs app is another variation of the HelloToast app you created in Lesson 1. It includes buttons to increment the number, to change the background color, and to reset both the number and color to their defaults. The app also uses themes and styles to define the buttons.



You start with the starter app and add shared preferences to the main activity code. You also add a reset button that sets both the count and the background color to the default, and clears the preferences file.

Task 1: Explore HelloSharedPrefs

The complete starter app project for this practical is available at HelloSharedPrefs-Starter. In this task you load the project into Android Studio and explore some of the app's key features.

1.1 Open and run the HelloSharedPrefs project

1. Download the HelloSharedPrefs-Starter app and unzip the file.
2. Open the project in Android Studio, and build and run the app. Try these things:
   - Click the **Count** button to increment the number in the main text view.
   - Click any of the color buttons to change the background color of the main text view.
   - Rotate the device and note that both background color and count are preserved.
   - Click the **Reset** button to set the color and count back to the defaults.
3. Force-quit the app using one of these methods:
   - In Android Studio, select **Run > Stop 'app'** or click the Stop Icon ■ in the toolbar.
   - On the device, press the Recents button (the square button in the lower right corner). Swipe the HelloSharedPrefs app card to quit the app, or click the X in the right corner of the card. If you quit the app in this manner, wait a few seconds before starting it again so the system can clean up.
4. Re-run the app. The app restarts with the default appearance—the count is 0, and the background color is grey.

1.2 Explore the Activity code

1. Open `MainActivity`.
2. Examine the code and note these things:
    - The count (`mCount`) is defined as an integer. The `countUp()` onClick method increments this value and updates the main `TextView`.
    - The color (`mColor`) is also an integer that is initially defined as grey in the `colors.xml` resource file as `default_background`.
    - The `changeBackground()` onClick method gets the background color of the button that was clicked, and then sets the main text view to that color.
    - Both the `mCount` and `mColor` integers are saved to the instance state bundle in `onSaveInstanceState()`, and restored in `onCreate()`. The bundle keys for count and color are defined by private variables (`COUNT_KEY`) and (`COLOR_KEY`).

Task 2: Save and restore data to a shared preferences file

In this task you save the state of the app to a shared preferences file, and read that data back in when the app is restarted. Because the state data that you save to the shared preferences (the current count and color) are the **same** data that you preserve in the instance state, you don't have to do it twice. You can replace the instance state altogether with the shared preference state.

2.1 Initialize the preferences

1. Add member variables to the `MainActivity` class to hold the name of the shared preferences file, and a reference to a `SharedPreferences` object.

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

You can name your shared preferences file anything you want to, but conventionally it has the same name as the package name of your app.

2. In the `onCreate()` method, initialize the shared preferences. Insert this code before the `if` statement:

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

The `getSharedPreferences()` method (from the activity `Context`) opens the file at the given filename (`sharedPrefFile`) with the mode `MODE_PRIVATE`.

Note: 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, consider using content URIs provided by a `FileProvider`.

Solution code for `MainActivity`, partial:

```
public class MainActivity extends AppCompatActivity {
   private int mCount = 0;
   private TextView mShowCount;
   private int mColor;

   private SharedPreferences mPreferences;
   private String sharedPrefFile =
      "com.example.android.hellosharedprefs";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mShowCount = (TextView) findViewById(R.id.textview);
       mColor = ContextCompat.getColor(this,
          R.color.default_background);
       mPreferences = getSharedPreferences(
          sharedPrefFile, MODE_PRIVATE);

       // …
    }
}
```

2.2 Save preferences in onPause()

Saving preferences is a lot like saving the instance state -- both operations set aside the data to a Bundle object as a key/value pair. For shared preferences, however, you save that data in the `onPause()` lifecycle callback, and you need a shared editor object (`SharedPreferences.Editor`) to write to the shared preferences object.

1. Add the `onPause()` lifecycle method to `MainActivity`.

```
@Override
protected void onPause(){
   super.onPause();

   // ...
}
```

2. In `onPause()`, get an editor for the `SharedPreferences` object:

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

   A shared preferences editor is required to write to the shared preferences object. Add this line to `onPause()` after the call to `super.onPause()`.

3. Use the `putInt()` method to put both the `mCount` and `mColor` integers into the shared preferences with the appropriate keys:

```
preferencesEditor.putInt(COUNT_KEY, mCount);
preferencesEditor.putInt(COLOR_KEY, mColor);
```

The `SharedPreferences.Editor` class includes multiple "put" methods for different data types, including `putInt()` and `putString()`.

4. Call `apply()` to save the preferences:

```
preferencesEditor.apply();
```

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.

1. Delete the entire `onSaveInstanceState()` method. Because the activity instance state contains the same data as the shared preferences, you can replace the instance state altogether.

Solution code for `MainActivity` `onPause()` method:

```
@Override
protected void onPause(){
   super.onPause();

   SharedPreferences.Editor preferencesEditor = mPreferences.edit();
   preferencesEditor.putInt(COUNT_KEY, mCount);
   preferencesEditor.putInt(COLOR_KEY, mColor);
   preferencesEditor.apply();
}
```

2.3 Restore preferences in onCreate()

As with the instance state, your app reads any saved shared preferences in the `onCreate()` method. Again, because the shared preferences contain the same data as the instance state, we can replace the state with the preferences here as well. Every time `onCreate()` is called -- when the app starts, on configuration changes -- the shared preferences are used to restore the state of the view.

1. Locate the part of the `onCreate()` method that tests if the `savedInstanceState` argument is null and restores the instance state:

```
if (savedInstanceState != null) {
   mCount = savedInstanceState.getInt(COUNT_KEY);
   if (mCount != 0) {
       mShowCountTextView.setText(String.format("%s", mCount));
   }
   mColor = savedInstanceState.getInt(COLOR_KEY);
   mShowCountTextView.setBackgroundColor(mColor);
}
```

2. Delete that entire block.
3. In the `onCreate()` method, in the same spot where the instance state code was, get the count from the preferences with the `COUNT_KEY` key and assign it to the `mCount` variable.

```
mCount = mPreferences.getInt(COUNT_KEY, 0);
```

When you read data from the preferences you don't need to get a shared preferences editor. Use any of the "get" methods on a shared preferences object (such as `getInt()` or `getString()` to retrieve preference data.

Note that the `getInt()` method takes two arguments: one for the key, and the other for the default value if the key cannot be found. In this case the default value is 0, which is the same as the initial value of `mCount`.

4. Update the value of the main `TextView` with the new count.

```
mShowCountTextView.setText(String.format("%s", mCount));
```

5. Get the color from the preferences with the `COLOR_KEY` key and assign it to the `mColor` variable.

```
      mColor = mPreferences.getInt(COLOR_KEY, mColor);
```

As before, the second argument to `getInt()` is the default value to use in case the key doesn't exist in the shared preferences. In this case you can just reuse the value of `mColor`, which was just initialized to the default background further up in the method.

6. Update the background color of the main text view.

```
mShowCountTextView.setBackgroundColor(mColor);
```

7. Run the app. Click the **Count** button and change the background color to update the instance state and the preferences.
8. Rotate the device or emulator to verify that the count and color are saved across configuration changes.
9. Force-quit the app using one of these methods:
   - In Android Studio, select **Run > Stop 'app.'**
   - On the device, press the Recents button (the square button in the lower right corner). Swipe the HelloSharedPrefs app card to quit the app, or click the X in the right corner of the card.
10. Re-run the app. The app restarts and loads the preferences, maintaining the state.

Solution code for `MainActivity` `onCreate()` method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   // Initialize views, color, preferences
   mShowCountTextView = (TextView) findViewById(R.id.count_textview);
   mColor = ContextCompat.getColor(this, R.color.default_background);
   mPreferences = getSharedPreferences(
                       mSharedPrefFile, MODE_PRIVATE);

   // Restore preferences
   mCount = mPreferences.getInt(COUNT_KEY, 0);
   mShowCountTextView.setText(String.format("%s", mCount));
   mColor = mPreferences.getInt(COLOR_KEY, mColor);
   mShowCountTextView.setBackgroundColor(mColor);
}
```

2.4 Reset preferences in the reset() click handler

The reset button in the starter app resets both the count and color for the activity to their default values. Because the preferences hold the state of the activity, it's important to also clear the preferences at the same time.

1. In the `reset()` onClick method, after the color and count are reset, get an editor for the `SharedPreferences` object:

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

2. Delete all the shared preferences:

```
preferencesEditor.clear();
```

1. Apply the changes:

```
preferencesEditor.apply();
```

Solution code for the `reset()` method:

```
public void reset(View view) {
    // Reset count
    mCount = 0;
    mShowCountTextView.setText(String.format("%s", mCount));

    // Reset color
    mColor = ContextCompat.getColor(this, R.color.default_background);
    mShowCountTextView.setBackgroundColor(mColor);
```

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

}
```