Persist Login State

This lesson will cover how to store login state in the Android key-value storage.

WE'LL COVER THE FOLLOWING

- Flow overview
- Shared preferences
- Travel blog preferences

Flow overview

When *LoginActivity* is opened, we need to check the login state:

- If the user is logged in already, we close *LoginActivity* and open *MainActivity* straightaway.
- If the user is not logged in, we proceed to the regular flow and save the login state in the end.

Shared preferences

In Android, to store a simple key-value data `SharedPreferences` can be used. This storage support saving the following types of data:

- `String`
- `boolean`
- `int`
- `long`
- `float`
- `Set<String> values`

Any data stored in the `SharedPreferences` is going to be persisted inside the internal application-specific file on the file system. Let's learn how to create the `SharedPreferences` and some basic operations.

the `SharedPreferences` and some basic operations.

To create new or load existing `SharedPreferences`, we can use `Context#getSharedPreferences` method. Don't forget that the `Activity` class implements the `Context` interface, so this method can be called inside the `Activity`.

Method `getSharedPreferences` has two required parameters:

- name - the shared preferences file name, in our case `travel-blog`
- mode - the operating mode, in our case `Context.MODE_PRIVATE` means that only our application will have access to this shared preferences

```
SharedPreferences preferences
      = context.getSharedPreferences("travel-blog", Context.MODE_PRIVATE);
```

Example

To retrieve the data from shared preferences, one of the available shared preferences `get` method can be used. Every get method has two required parameters:

- key - the name of the preference, in our case `key_login_state`
- default value - the default value to be returned if data is not available, in our case `false`

```
preferences.getBoolean("key_login_state", false);
```

Sample

To store the data into shared preferences, we can use the `SharedPreferences#edit` method which returns the `Editor` object. This object has put methods to store different types of data. Every put method has two required parameters:

- key - the name of the preference, in our case `key_login_state`
- value - the value which we want to store, in our case `true`

In the end, we have to call the `apply` method which is going to store data into the memory and asynchronously persist on the file system.

```
preferences.edit().putBoolean("key_login_state", true).apply();
```

Example

Travel blog preferences

Now that we have learned how to work with shared preferences, let's use them to store a boolean flag to indicate whether the user is logged in or not.

Instead of adding code directly into *LoginActivity*, it's a good idea to move the logic of storing login state into a separate file.

Let's create a `BlogPreferences` class and add all the related logic there:

- in the constructor, we create shared preferences
- `isLoggedIn` method retrieves the value from shared preferences
- `setLoggedIn` method sets the value into shared preferences

```
public class BlogPreferences {

    private static final String KEY_LOGIN_STATE = "key_login_state";

    private SharedPreferences preferences;

    BlogPreferences(Context context) {
        preferences =
            context.getSharedPreferences("travel-blog", Context.MODE_PRIVATE);
    }

    public boolean isLoggedIn() {
        return preferences.getBoolean(KEY_LOGIN_STATE, false);
    }

    public void setLoggedIn(boolean loggedIn) {
        preferences.edit().putBoolean(KEY_LOGIN_STATE, loggedIn).apply();
    }
}
```

BlogPreferences

It's time to use *BlogPreferences* in the *LoginActivity*. Let's create `BlogPreferences` and add our check inside the `onCreate` method.

It's a good idea to do that before executing the `setContentView` method since if the user is logged in, we need to open *MainActivity* and finish the current activity, without rendering user interface at all.

The `return` at the end of `if` statement makes sure that following code is not going to be executed.

```
public class LoginActivity extends AppCompatActivity {
    ...
    private BlogPreferences preferences;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        preferences = new BlogPreferences(this);
        if (preferences.isLoggedIn()) {
            startMainActivity();
            finish();
            return;
        }

        setContentView(R.layout.activity_login);
        ...
    }
...
}
```

LoginActivity

Finally, we can change the login state flag in the `performLogin` method, before opening *MainActivity*.

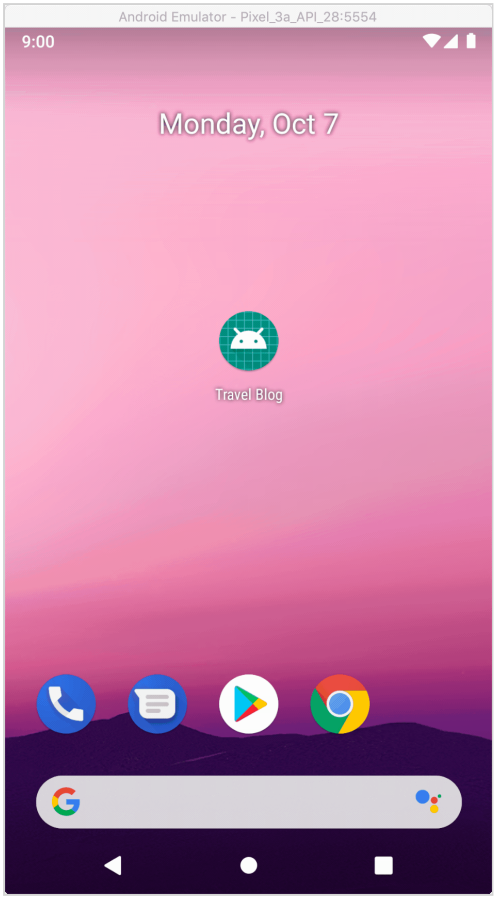
```
private void performLogin() {
    preferences.setLoggedIn(true);

    textUsernameLayout.setEnabled(false);
    textPasswordInput.setEnabled(false);
    loginButton.setVisibility(View.INVISIBLE);
    progressBar.setVisibility(View.VISIBLE);

    new Handler().postDelayed(() -> {
        startMainActivity();
        finish();
    }, 5000);
}
```

LoginActivity

As you can see in the preview below, after we successfully logged the first time, all the consequential application launches lead us straight to the *MainActivity*.



Hit the *run* button to try it yourself.

```
package com.travelblog;

import android.content.Context;
import android.content.SharedPreferences;

public class BlogPreferences {

    private static final String KEY_LOGIN_STATE = "key_login_state";

    private SharedPreferences preferences;

    BlogPreferences(Context context) {
        preferences =
                context.getSharedPreferences("travel-blog", Context.MODE_PRIVATE);
    }

    public boolean isLoggedIn() {
        return preferences.getBoolean(KEY_LOGIN_STATE, false);
    }

    public void setLoggedIn(boolean loggedIn) {
        preferences.edit().putBoolean(KEY_LOGIN_STATE, loggedIn).apply();
    }
}
```

In the next lesson, we will cover how to apply custom styles.