**LAB # 12**

**Persisting Data – SharedPreferences**

Shared Preferences is a mechanism provided by the Android framework to store and retrieve small amounts of data in key-value pairs. It allows you to persistently store data in a private file associated with your application. Shared Preferences is commonly used for storing app settings, user preferences, and other lightweight data that needs to be accessed across multiple sessions of your application.

Here are some key features and characteristics of Shared Preferences:

1. Lightweight storage: Shared Preferences is designed for storing small amounts of data, such as primitive values (integers, booleans, floats, etc.), strings, and sets of strings. It is not suitable for storing large data sets or complex objects.
2. Private storage: Each Android application has its own isolated set of Shared Preferences files. The data stored in Shared Preferences is private to your application and cannot be accessed by other applications unless explicitly shared.
3. Key-value pairs: Data in Shared Preferences is organized as key-value pairs. You store a value with a unique key and retrieve it using the same key.
4. Persistence: Data stored in Shared Preferences is persistent, meaning it remains available even when the application is closed or the device is restarted.
5. Simple API: Shared Preferences provides a simple and straightforward API for reading, writing, updating, and removing data. It abstracts the complexities of file I/O and data serialization.
6. Mode options: When creating a Shared Preferences file, you can specify the access mode. The most common mode is Context.MODE\_PRIVATE, which ensures that only your application can access the preferences. Other modes, such as Context.MODE\_WORLD\_READABLE and Context.MODE\_WORLD\_WRITEABLE, allow limited access to other applications.

Shared Preferences is a convenient way to store and retrieve persistent data in Android applications. It provides an efficient and easy-to-use solution for managing user preferences, app settings, and other small data sets.

Step 1: **Create a new Android project**

Start by creating a new Android project in Android Studio or your preferred IDE. Set up your project with the necessary dependencies and resources.

Step 2: **Working with Shared Preferences**

To use Shared Preferences, follow these steps:

1. **Obtain a reference to the Shared Preferences object:**

Open your desired activity or fragment file where you want to use Shared Preferences. Inside the class, add the following code:

SharedPreferences sharedPreferences = getSharedPreferences("MyPreferences", Context.MODE\_PRIVATE);

The first parameter, "MyPreferences," is the name of the preferences file. You can use any name you like. The second parameter, `Context.MODE\_PRIVATE`, specifies that only your app can access the preferences.

2. **Writing data to Shared Preferences:**

Inside the same activity or fragment file, add the following code to write data to Shared Preferences:

SharedPreferences.Editor editor = sharedPreferences.edit();

editor.putString("username", "John");

editor.putInt("age", 25);

editor.putBoolean("isPremium", true);

editor.apply(); // Or editor.commit() for immediate write, but apply() is generally preferred

In this example, we store a username as a String, an age as an integer, and a boolean value indicating if the user is a premium user.

3. **Reading data from Shared Preferences:**

To retrieve data from Shared Preferences, you can use the various `getX()` methods available, such as `getString()`, `getInt()`, `getBoolean()`, etc. Inside the same activity or fragment file, add the following code to read data from Shared Preferences:

String username = sharedPreferences.getString("username", "");

int age = sharedPreferences.getInt("age", 0);

boolean isPremium = sharedPreferences.getBoolean("isPremium", false);

In this example, we retrieve the stored username, age, and premium status. The second parameter in each method is the default value to return if the preference doesn't exist.

4. **Updating and removing data:**

Inside the same activity or fragment file, you can update or remove data from Shared Preferences using the following code:

To update data:

SharedPreferences.Editor editor = sharedPreferences.edit();

editor.putString("username", "NewUsername");

editor.apply();

This example updates the username to "NewUsername" in Shared Preferences.

To remove data:

SharedPreferences.Editor editor = sharedPreferences.edit();

editor.remove("username");

editor.apply();

This example removes the username from Shared Preferences.

5. **Clearing all data:**

If you want to clear all data from the Shared Preferences file, you can use the following code inside the same activity or fragment file:

SharedPreferences.Editor editor = sharedPreferences.edit();

editor.clear();

editor.apply();

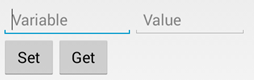
```

This example removes all data from the preferences file.

Remember to handle exceptions appropriately when working with Shared Preferences and consider security concerns for sensitive data.

**TASKS**

1. Saving customized key-value pair: Create an app by which user can set and get customized key-value pair. You may follow the GUI given below:



1. Create an app which can save user comments recorder by using SharedPreferences. User must be able to see previous comments chronologically.

