ENCS5150 · EXP 6 COMPANION

# Singletons and Shared Preferences

One object for the whole app, a small key–value file that survives restarts, what must never go in it, and what replaces it today.

# Five questions

01 What is a singleton, and when is it worth it?

02 How do SharedPreferences store data?

03 apply() or commit()?

04 What must never be stored there?

05 What do modern apps use instead?

01 · THE SINGLETON

# One object, three parts

```
public class SharedPrefManager {
    private static SharedPrefManager instance = null;

    private SharedPrefManager(Context context) { … }

    public static synchronized SharedPrefManager
            getInstance(Context context) {
        if (instance == null) {
            instance = new SharedPrefManager(context);
        }
        return instance;
    }
}
```

**1 · STATIC FIELD**

Belongs to the class, so there is one for the whole app.

**2 · PRIVATE CONSTRUCTOR**

new SharedPrefManager() outside the class doesn’t compile.

**3 · getInstance**

Creates the object on the first call, returns the same one after: lazy initialization.

01 · THE SINGLETON

# Two details that make it safe

SYNCHRONIZED

## Only one thread at a time

| | |
|---|---|
| Thread A: instance is null | Thread B: instance is null |
| A creates an object | B creates another |

Without **synchronized**, two threads arriving together can each make one. With it, B waits, then finds A’s object.

THE RIGHT CONTEXT

## Never keep an Activity

A singleton lives as long as the app. If it kept a reference to the Activity passed in, that Activity could never be freed: a memory leak.

The lab’s manager uses the context once to open the file and keeps only the SharedPreferences. When a singleton must keep a context, store **context.getApplicationContext()**.

01 · THE SINGLETON

# When a singleton is worth it, and what it costs

GOOD FOR

## One shared resource

Something expensive to create twice or that must be shared: a database helper (Exp 4), a network client, a settings file.

One class owns the file name and the keys.

COSTS

## Global state

Any class can reach it and change it, so bugs are hard to trace.

Tests can’t swap in a fake, and it holds memory until the app dies.

Larger apps get the same “one shared object” from **dependency injection** (Hilt): the object is created once and handed to whoever needs it, without a static field.

02 · SHARED PREFERENCES

# A key–value map, saved as an XML file

/data/data/<package>/shared_prefs/exp6_prefs.xml

```
<map>
    <boolean name="remember_password" value="true" />
    <string name="password">secret123</string>
    <int name="save_count" value="2" />
    <string name="username">ahmad</string>
</map>
```

## Write through an Editor

```
prefs.edit()
  .putInt("save_count", 3)
  .apply();
```

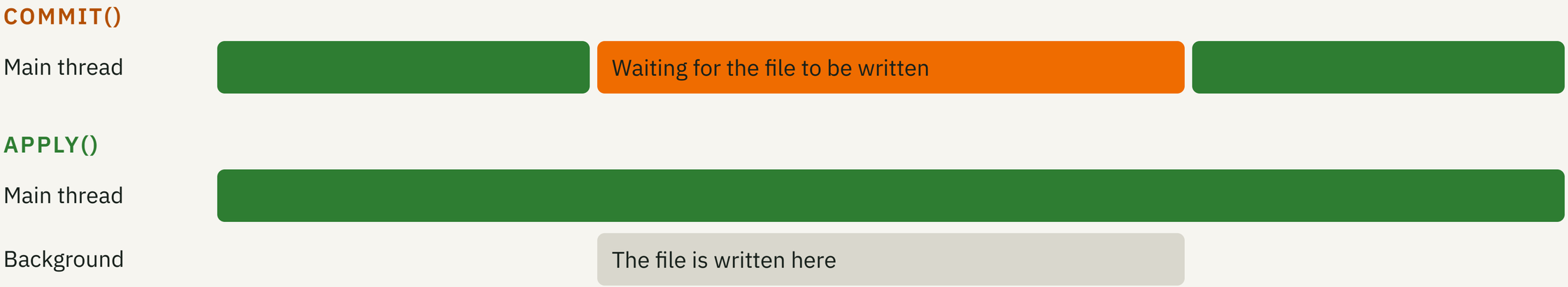
## Read with a default

```
prefs.getInt("save_count", 0);
```

Types: String, int, long, float, boolean and Set<String>. No double, byte or char, and no objects.

02 · SHARED PREFERENCES

# apply() returns at once; commit() waits for the disk




Both update the values in memory immediately, so the next read sees them. Use **apply()** unless you need commit()’s true or false.

# Preferences or a database?

| | Shared preferences | SQLite or Room (Exp 4) |
|---|---|---|
| Holds | A few single values | Many rows of the same shape |
| Example | Last username, dark mode on, a counter | A list of customers |
| Look up by | Key | Any WHERE condition |
| Loaded | The whole file, into memory | Only the rows you ask for |

If you’re tempted to store a list in preferences, you want a database.

03 · WHAT NEVER GOES THERE

# Never store a password in preferences

IT IS PLAIN TEXT

### Readable by anyone who gets the file

A rooted phone, a debug build, or a device backup: the template’s allowBackup="true" includes shared_prefs.

WHAT APPS DO INSTEAD

### Send it once, keep a token

The app sends the password to the server at login, and stores only the token the server sends back. The token can expire and be revoked.

The lab saves one only so you can open the file and see this for yourself.

03 · IS THERE SOMETHING BETTER?

# Jetpack DataStore replaces SharedPreferences

| | SharedPreferences | DataStore |
|---|---|---|
| Reads and writes | Can block the main thread | Always asynchronous |
| Errors | Silently ignored | Reported to your code |
| Saving | apply() or commit(), easy to get wrong | One transactional update |
| Kinds | Key–value only | Key–value, or typed with a schema (Proto) |
| Language | Java or Kotlin | Kotlin coroutines and Flow; RxJava for Java |

Google’s guidance: use DataStore for new code, and consider migrating existing SharedPreferences to it. The lab uses SharedPreferences because its API fits plain Java.

TAKEAWAYS

# Four things to remember

01 A singleton is a static field, a private constructor and a synchronized getInstance.

02 Never keep an Activity in it: use the application context.

03 Preferences hold a few small values: apply(), keys as constants, never passwords.

04 New apps use DataStore for settings and Room for data.