RV College of Engineering®

Department of Master of Computer Applications

Mobile Application Development

Storing Simple Data

Selvadiwakar . G (1RV24MC097)

Under the Guidance of Deepika Kripanithi

Go, change the world®

Introduction to Data Storage in Mobile Apps

- **Why Store Data?**
  Persistence: Data remains even after app closes or device restarts.
  Enables features like user preferences, offline access, caching, saving app state.

- **Categories of Data Storage:**
  Ranges from simple key-value pairs to complex databases.
  Choice depends on data complexity, size, and access patterns.
  Focus Today: Methods for storing simple, non-structured, small-volume data.

Shared Preferences: Overview

- **Definition:** Android mechanism for storing private primitive data in key-value pairs.

- **Ideal For:** Simple settings, user preferences, small configuration data.

- **Key Characteristics:**
  Private: App-specific, generally not accessible by other apps.
  Primitive Data Types: boolean, float, int, long, String.
- Lightweight: Efficient for small data volumes.
- Internal Storage: Stored internally in XML files.

- **When to Use:**
  Dark mode preference, notification settings.
  "Remember Me" login status.
  High scores in a game.

RV College of Engineering®

Shared Preferences: How to Use (Writing)

Go, change the world®

1) Get SharedPreferences Instance:

getSharedPreferences(name, mode): For named, multiple files.

getPreferences(mode): For Activity-specific, single file.

(Typically use Context.MODE_PRIVATE)

2) Get an Editor Object:

SharedPreferences.Editor editor = mySharedPreferences.edit();

3) Put Data (Key-Value Pairs):

editor.putBoolean("key_name", true);

editor.putString("user_name", "Alice");

editor.putInt("score", 100);

(and others for float, long)

4)Apply Changes:

editor.apply();

Asynchronous write (recommended for UI thread safety).

editor.commit();

Synchronous write (returns true on success; avoid on UI thread if possible).

RV College of Engineering®

Shared Preferences: How to Use (Reading)

Go, change the world®

1) Get SharedPreferences Instance (same as for writing).

2) Read Data (Specify Default Value):

- boolean isDarkTheme = mySharedPreferences.getBoolean("is_dark_theme", false);
- String savedUserName = mySharedPreferences.getString("user_name", "Guest");
- int highScore = mySharedPreferences.getInt("score", 0);
- (and others)
- Default Value: Returned if the key is not found.

Shared Preferences: Example

```
// Get SharedPreferences instance (e.g., in an Activity)
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

// --- WRITING DATA ---
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean("is_dark_theme", true); // Save user's theme preference
editor.apply(); // Asynchronously save changes

// --- READING DATA ---
boolean isDarkTheme = sharedPref.getBoolean("is_dark_theme", false);
// 'isDarkTheme' will be true if saved, false otherwise
```

RV College of Engineering®

Internal Storage (Files): Overview

Go, change the world®

- **Definition:** Allows storing files directly on the device's private internal memory.

- Key Characteristics:
  1) **Private to App:** Files are typically inaccessible to other apps and the user.
  2) **Automatic Deletion**: Files are removed when your app is uninstalled.
  3)**Limited Space**: Can be constrained by device storage.

- When to Use:
  1) Storing app-specific configuration files.
  2) Caching sensitive data (if encryption is handled separately).
  3) Saving user-generated content relevant only to your app (e.g., small custom profiles).
  4) Simple log files.

RV College of Engineering®

Internal Storage (Files): How to Use (Writing)

Go, change the world®

- Get File Output Stream:

```
FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
```

  FILENAME: Name of your file (e.g., "my_data.txt").
  MODE_PRIVATE: File is private to your app.
  MODE_APPEND: Appends to existing file.

- Write Data:

```
fos.write("Your text or bytes here".getBytes());
```

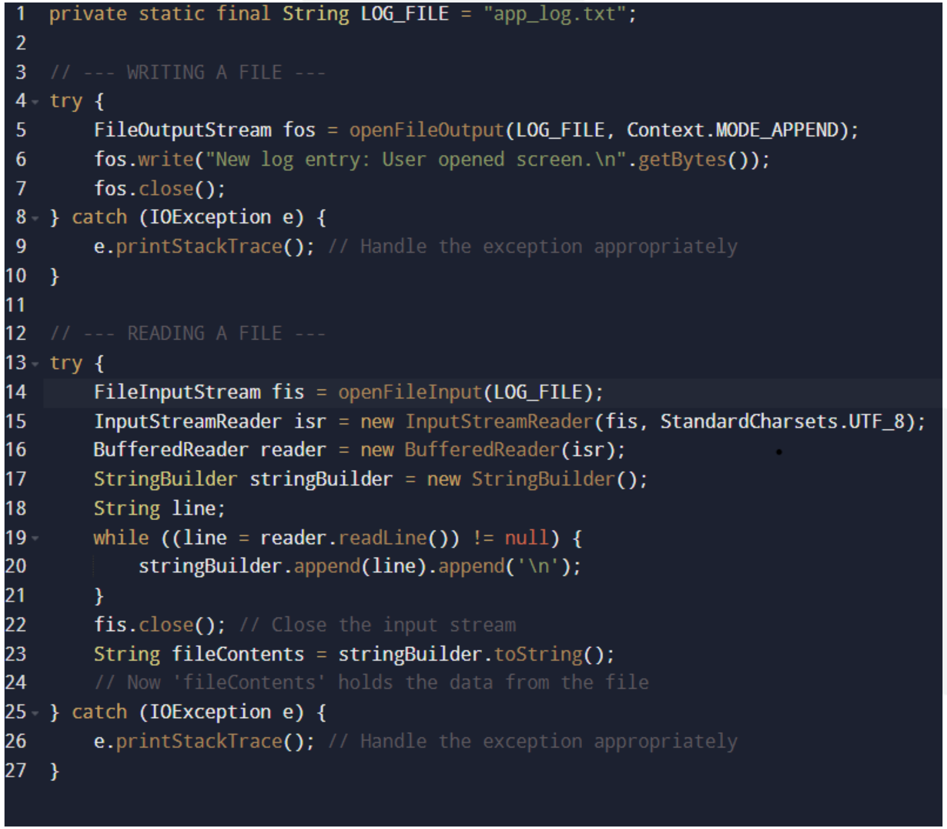
- Close Stream:

```
fos.close();
```

- Error Handling:

  Crucial: Always wrap file I/O operations in a try-catch (IOException e) block.

Internal Storage (Files): Example

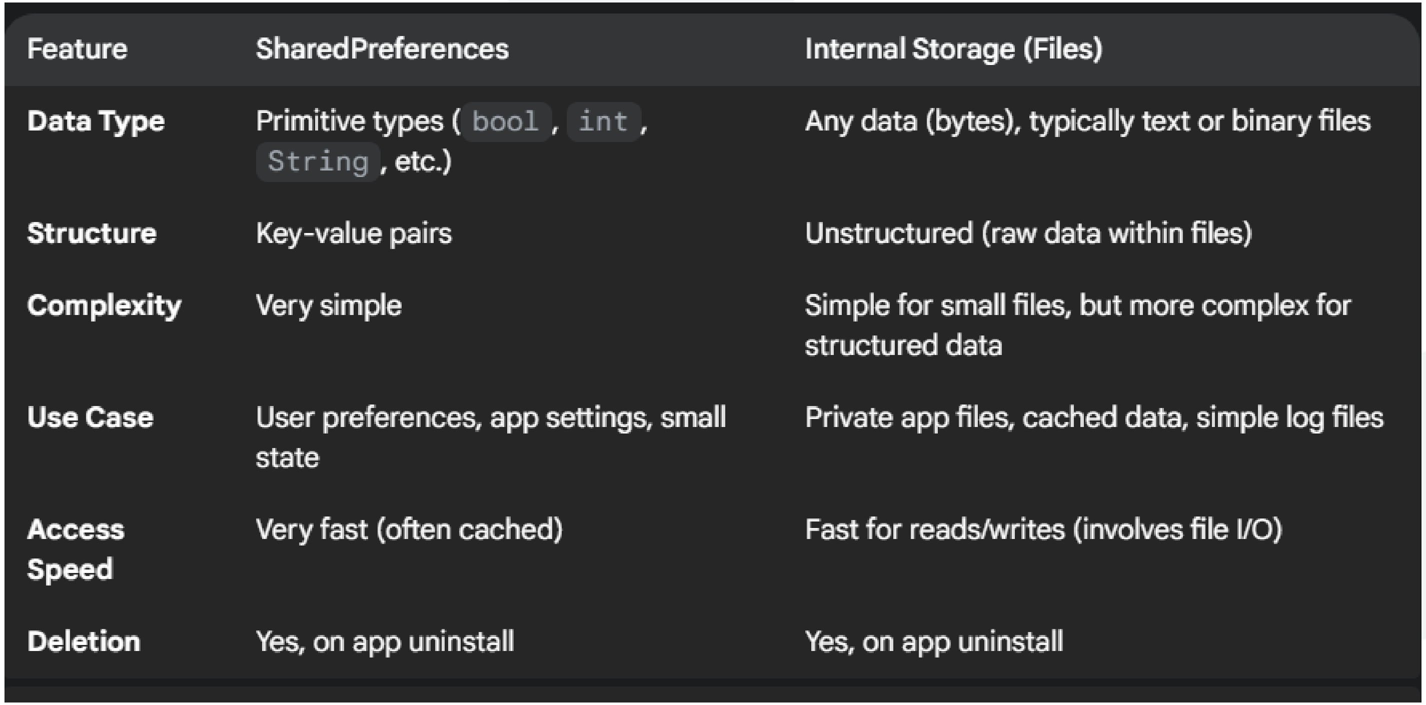


```java
private static final String LOG_FILE = "app_log.txt";

// --- WRITING A FILE ---
try {
    FileOutputStream fos = openFileOutput(LOG_FILE, Context.MODE_APPEND);
    fos.write("New log entry: User opened screen.\n".getBytes());
    fos.close();
} catch (IOException e) {
    e.printStackTrace(); // Handle the exception appropriately
}

// --- READING A FILE ---
try {
    FileInputStream fis = openFileInput(LOG_FILE);
    InputStreamReader isr = new InputStreamReader(fis, StandardCharsets.UTF_8);
    BufferedReader reader = new BufferedReader(isr);
    StringBuilder stringBuilder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        stringBuilder.append(line).append('\n');
    }
    fis.close(); // Close the input stream
    String fileContents = stringBuilder.toString();
    // Now 'fileContents' holds the data from the file
} catch (IOException e) {
    e.printStackTrace(); // Handle the exception appropriately
}
```

RV College of Engineering®

Comparison: Shared Preferences vs. Internal Files

Go, change the world®



Feature	SharedPreferences	Internal Storage (Files)
Data Type	Primitive types (`bool` , `int` , `String` , etc.)	Any data (bytes), typically text or binary files
Structure	Key-value pairs	Unstructured (raw data within files)
Complexity	Very simple	Simple for small files, but more complex for structured data
Use Case	User preferences, app settings, small state	Private app files, cached data, simple log files
Access Speed	Very fast (often cached)	Fast for reads/writes (involves file I/O)
Deletion	Yes, on app uninstall	Yes, on app uninstall

Best Practices for Simple Data Storage

1) Choose the Right Tool:
- SharedPreferences for small, non-structured key-value settings.
- Internal Files for app-specific, private data (text, small binary files).

2) Security:
- Neither offers strong encryption by default. For sensitive data, consider encryption libraries.

3) UI Thread Safety:
- File I/O (Internal Storage): ALWAYS perform on a background thread to prevent ANRs.
- SharedPreferences.apply(): Safe for UI thread (asynchronous).
- SharedPreferences.commit(): Avoid on UI thread (synchronous).

4) Error Handling:
- Always use try-catch blocks for IOException when working with file operations.