Platform-Based Development: Data Storage

BS UNI studies, Spring 2019/2020

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Partly based on “Programming Handheld Systems”,
Adam Porter, University of Maryland

Data Storage

- SharedPreferences
- Internal storage
- External storage
- SQLite database

SharedPreference

- Preserve a small amount of primitive type (int, float, String, Boolean) data on a device
  - Data are saved as key-value pairs
  - Should be read/written by your app only
  - Stored for as long as the app is installed on a device
- Common use:
  - User preferences – username, customisations, such as preferred WiFi AP, preferred theme, etc.
  - Variables for conditional app execution
    - When the app is launched for the first time set “launched” to True; next time, check if “launched” was set or not

http://developer.android.com/reference/android/content/SharedPreferences.html

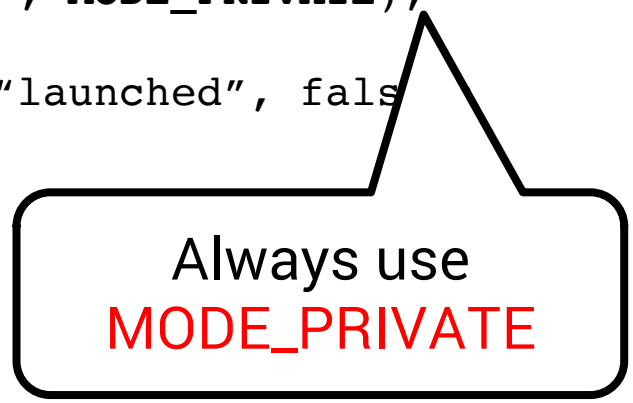
Accessing SharedPreferences

- Reading

```
SharedPreferences settings = getApplicationContext()
        .getSharedPreferences("preferences", MODE_PRIVATE);

boolean wasLaunched = settings.getBoolean("launched", fals
```

- Need to know the type of data:
  - getBoolean()
  - getString()
  - getAll() returns a Map of key-value pairs



Accessing SharedPreferences

- Writing

```
SharedPreferences settings = getApplicationContext()
        .getSharedPreferences("preferences", MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("launched", true);
editor.commit();
```

- Different put methods for different data types
- Don't forget to save changes by calling
  - editor.commit() – synchronous (avoid calling on the main thread) or
  - editor.apply() – changes the in-memory object immediately, but writes to disk asynchronously

PreferenceFragmentCompat

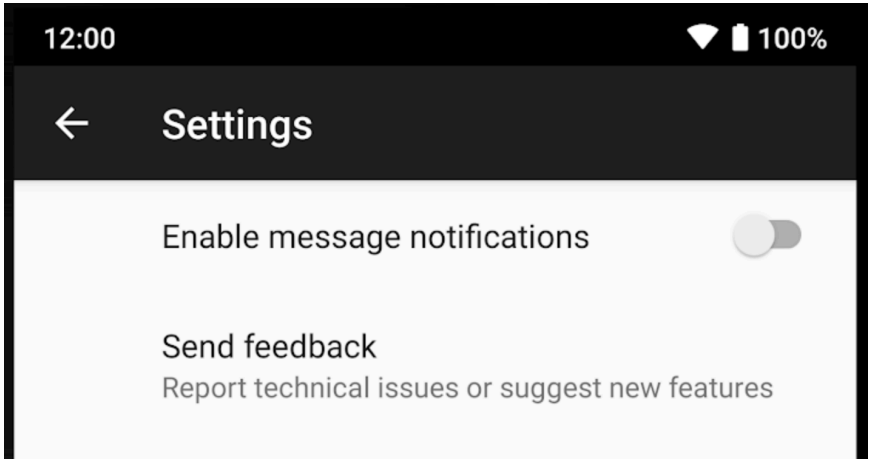
- A special Fragment that connects XML preference hierarchy with SharedPreferences

```
<androidx.preference.PreferenceScreen
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <SwitchPreferenceCompat
        app:key="notifications"
        app:title="Enable message
                    notifications"/>

    <Preference
        app:key="feedback"
        app:title="Send feedback"
        app:summary="Report technical
       issues or suggest new features"/>

</androidx.preference.PreferenceScreen>
```



https://developer.android.com/guide/topics/ui/settings.html

PreferenceFragmentCompat

- Inflate the hierarchy in a Fragment (PreferenceFragmentCompat):

```
public class MySettingsFragment extends PreferenceFragmentCompat {
    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String
rootKey) {
        setPreferencesFromResource(R.xml.preferences, rootKey);
    }
}
```

- Note: a part of Android X, don't confuse with the same class from the old support library or with PreferenceFragment – a deprecated class

https://developer.android.com/guide/topics/ui/settings.html

File Storage

- Android uses the common Java File API
- Files saved as internal or external files

INTERNAL	EXTERNAL
Always available.	Not always available – a user can mount the external storage as USB storage
Files saved here are accessible by only your app by default.	World-readable, so files saved here may be read outside of your control.
When the user uninstalls your app, the system removes all your app's files from internal storage.	When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from getExternalFilesDir().

Internal vs External

- What we have in mind:
  - Internal – on-device (non-removable) storage
  - External – SD card (removable)
- What Google has in mind:
  - Internal – a part of non-removable storage that is not shared among apps
  - External – either removable or non-removable storage that is shared among apps

Internal Files

- Find where to save files
  - getFilesDir() an internal directory for your app
  - getCacheDir() an internal directory for your app's temporary cache files
    - File createTempFile() – to create a unique filename
    - If the system begins running low on storage, it may delete these cache files
- Read/write methods
  - FileOutputStream openFileOutput (String name, int mode) opens file for writing (creates it if needed)
  - FileInputStream openFileInput (String name) opens a file for reading

External Files

- Public external files –accessible by other applications as well
  - Remain on the device even after your app is uninstalled
  - getExternalStoragePublicDirectory() to get a directory where these files live
  - Use Environment class constants for storing files in appropriate directories

```
File file = new File(Environment
            .getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES),albumName);
```

External Files

- Private external files –accessible by your application only
  - Removed when your application is uninstalled
  - getExternalFilesDir() to get a directory where these files live
  - Use Environment class constants for storing files in appropriate directories

```
File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName);
```

External Files

- Reading and writing to external storage requires permissions

```
<manifest ...>
 <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE”/>
</manifest>
```

- Before working with external storage, check if the storage is mounted

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

SQLite

- A relational database implementation for embedded systems
  - A library added to an app, rather than a standalone client-served DB engine
  - The whole database is kept in a single file, and a write operation gets an exclusive lock over the file (no concurrent writing)
  - SQL-92 language standard implemented (mostly)
  - Supports ACID transactions (Atomic, Consistent, Isolated and Durable)

When to Use SQLite

- Persistent storage (just like internal files)
- Use Android's SQLite database for structured data, especially if SQL queries can be used for improved data presentation
- Note: the database will be removed once your application is uninstalled

Working with SQLite

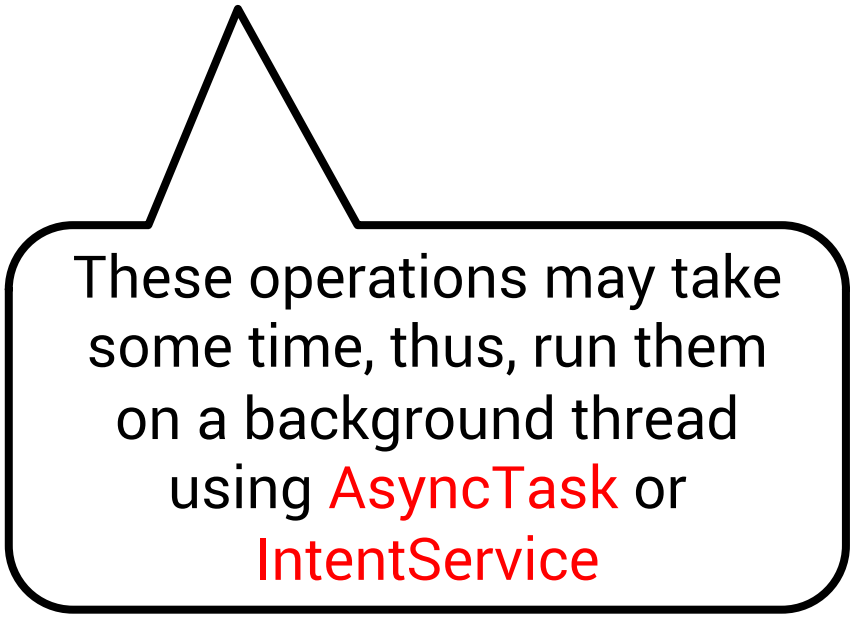
- Work directly via the following key classes:
  - SQLiteOpenHelper
    - To create, open, modify the tables in the database
  - SQLiteDatabase
    - To query the database
- Work indirectly through an Object Relational Mapping (ORM):
  - Room – Google's own ORM
  - Greendao
  - ORMlite

SQLiteOpenHelper

- Subclass SQLiteOpenHelper for your database
  - Define your schema - formal declaration of how the database is organized, and the SQL statements that you use to create your database
- Call super() from the subclass constructor to initialize the underlying database
- Execute database creation in onCreate()
  - onCreate() of the helper is called when a component tries to access a not-yet-existing database
  - SQLiteOpenHelper abstracts the costly DB creation operations, and does not run them when not needed

SQLiteOpenHelper

- Access the database via SQLiteOpenHelper:
  - getReadableDatabase()
  - getWritableDatabase()



SQLiteOpenHelper Example

```
public class DatabaseHelper extends SQLiteOpenHelper {

    final static String DATABASE_NAME = "contacts_db";
    final static Integer DATABASE_VERSION = 1;
    final static String TABLE_NAME = "contacts";
    final static String USER_NAME = "name";
    final static String USER_EMAIL = "email";
    final static String _ID = "_id";
    final static String[] columns = { _ID, USER_NAME, USER_EMAIL };

    final private static String CREATE_CMD =

            "CREATE TABLE "+TABLE_NAME+" (" + _ID
                    + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                    + USER_NAME + " TEXT NOT NULL,"
                    + USER_EMAIL + " TEXT NOT NULL)";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_CMD);
    }
```

Use _id for the primary key

SQLiteOpenHelper Example

- Use in Activity:

```
public class MainActivity extends AppCompatActivity {

    private DatabaseHelper mDbHelper;
    private SimpleCursorAdapter mAdapter;

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

        mDbHelper = new DatabaseHelper(this);

        // Empty database at the start
        mDbHelper.getWritableDatabase().delete(DatabaseHelper.TABLE_NAME, null, null);
```

SQLiteDatabase

- Insert into database by calling insert() with ContentValues key-value pairs

```
ContentValues values = new ContentValues();
values.put(DatabaseHelper.USER_NAME, "Veljko Pejovic");
values.put(DatabaseHelper.USER_EMAIL, "Veljko.Pejovic@fri.uni-lj.si");
mDbHelper.getWritableDatabase().insert(DatabaseHelper.TABLE_NAME, null, values);
```

- Query the database with query() and rawQuery()

```
Cursor c = db.query(
    FeedEntry.TABLE_NAME,  // The table to query
    projection,                               // The columns to return
    selection,                                // The columns for the WHERE clause
    selectionArgs,                            // The values for the WHERE clause
    null,                                     // don't group the rows
    null,                                     // don't filter by row groups
    sortOrder                                 // The sort order
    );
```

SQLiteDatabase

- ListViews populated with Adapters connected to the database are often the most convenient way of displaying DB content

```
Cursor c = readContacts();
mAdapter = new SimpleCursorAdapter(this, R.layout.list_layout, c,
        DatabaseHelper.columns, new int[] { R.id._id, R.id.name, R.id.email}, 0);
ListView list = (ListView) findViewById(R.id.list);
list.setAdapter(mAdapter);
```

Examining DB Content

- Databases stored in /data/data/<package name>/databases/
- Can examine database with sqlite3

```
# adb -s emulator-5554 shell
# sqlite3
/data/data/si.uni_lj.fri.lrss.databaseexample/data
bases/ contacts_db
```

Examining DB Content

```
sqlite> .dump contacts
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE contacts (_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL,email TEXT NOT NULL);
INSERT INTO "contacts" VALUES(6,'Veljko Pejovic','Veljko.Pejovic@fri.uni-lj.si');
INSERT INTO "contacts" VALUES(7,'Fabio Ricciato','Fabio.Ricciato@fri.uni-lj.si');
INSERT INTO "contacts" VALUES(8,'Tomaz Curk','Tomaz.Curk@fri.uni-lj.si');
INSERT INTO "contacts" VALUES(9,'Miha Mraz','Miha.Mraz@fri.uni-lj.si');
COMMIT;
```

Note, you might have to run the emulator as root (adb root)