ANDROID - Curs 3

Topics:

- VCS
- Shared Preferences
- Local databases
- AsyncTask
- JSON

Version Control System (VCS)

Version Control Repositories Solutions:

- **GitHub** - create you account here: https://github.com/join
- GitLab
- Bitbucket

Tools:

- Terminal
- SourceTree
- Android Studio

How to create a new repo from scratch:

- Create a directory to contain the project.
- Go into the new directory in terminal
- Type **git init**.
- Write some code.
- Type **git add .** to add all files.
- Type **git commit**.

How to create a new repo from existing project:

- Go into the directory containing the project in terminal
- cd <folder_path>
- Type **git init**
- Type **git add .** to add all of the relevant files

- You'll probably want to create a .gitignore file right away, to indicate all of the files you don't want to track. *Use **git add .gitignore**, too.
- Type **git commit -m "First commit"**

Note: You can get Android .gitignore body from here: https://www.toptal.com/developers/gitignore

How to connect your project to GitHub:

- Go to GitHub.
- Log in to your account.
- Click the new repository button in the top-right. You'll have an option there to initialize the repository with a README file. DON'T DO IT!
- Click the "Create repository" button.
- Now, follow the second set of instructions, "Push an existing repository…":
  - `$ git remote add origin git@github.com:username/new_repo`
  - `$ git push -u origin master`

! Practice: Add lab2 on GitHub

Shared Preferences

! Use **SharedPreferences APIs** if you have a relatively small collection of key- values that you'd like to save.

1. You can **create a new shared preference file or access an existing one** by calling one of these methods:

• **getSharedPreferences()** — Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any Context in your app.

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
```

```
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

- **getPreferences()** — Use this from an Activity if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name.

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

2. Write to SharedPreferences

A. create a SharedPreferences.Editor by calling edit() on your SharedPreferences;
B. pass the keys and values you want to write with methods such as **putInt()** and **putString()**
C. call **apply()** or **commit()** to save the changes.
    - apply() - changes the in-memory SharedPreferences object immediately but writes the updates to disk asynchronously;
    - commit() - to write the data to disk synchronously. **Avoid calling it from your main thread!!!**

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score_key), newHighScore);
editor.commit();
```

3. Read from SharedPreferences

A. call methods such as **getInt()** and **getString();**
B. provide the key for the value you want;
C. optionally a default value to return if the key isn't present.

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.integer.saved_high_score_default_key);
int highScore = sharedPref.getInt(getString(R.string.saved_high_score_key), defaultValue);
```

Local Databases

- SQLite: https://developer.android.com/training/data-storage/sqlite

- Realm: https://github.com/realm/realm-java https://realm.io/docs/kotlin/latest/

- Room: https://developer.android.com/training/data-storage/room

Save data in a local database using Room

Intro

Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally. The most common use case is to cache relevant pieces of data.

That way, when the device cannot access the network, the user can still browse that content while they are offline. Any user-initiated content changes are then synced to the server after the device is back online.

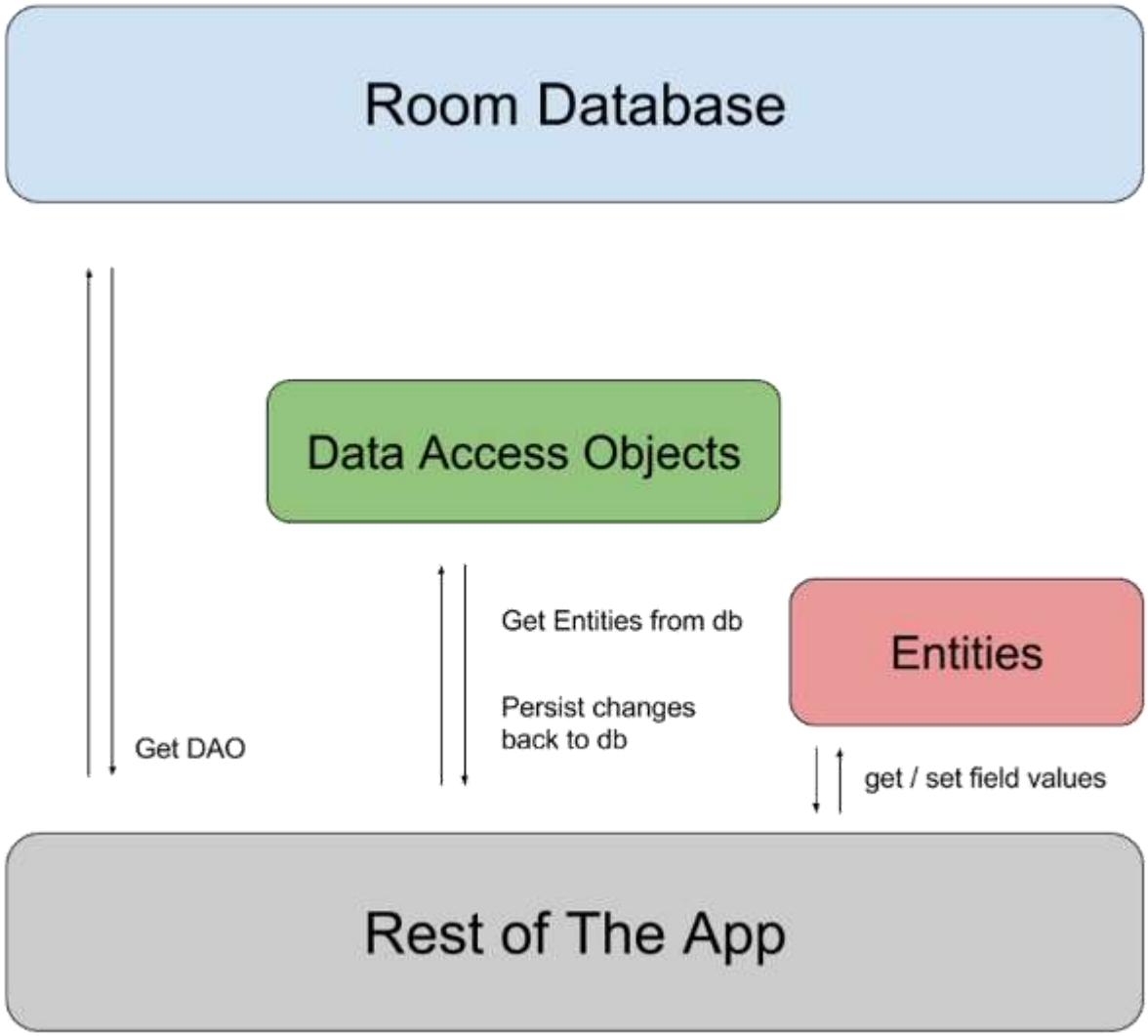
There are **3 major components i**n Room:

--> partea efectiva care contine datele

- **Database:** Contains the database holder and serves as the main access point for the underlying connection to your app's persisted, relational data.
- **Entity:** Represents a table within the database. --> reprezinta fiecare cate un tabel din baza de date
- **DAO:** Contains the methods used for accessing the database. --> contine metode prin care accesam baza de date

The app uses the Room database to get the **data access objects**, or DAOs, associated with that database. The app then uses each DAO to get entities from the database and save any changes to those entities back to the database.

Finally, the app uses an entity to get and set values that correspond to table columns within the database.



Implementation Steps

1. Migrate project to AndroidX (if necessary):

- Refactor -> Migrate to AndroidX…
- Clean and rebuild project

2. Add Room dependencies to gradle:

```
dependencies {
    def room_version = "2.1.0-alpha06"

    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
}
```

3. Create your entity following next model:

```
@Entity
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

4. Create your DAO following next model:

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

5. Create your database following next model:

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

6. Create your Application class:

```
//  If your app runs in a single process, you should follow the singleton design pattern
//  when instantiating an AppDatabase object.
    private static ApplicationController mInstance;

//  Each RoomDatabase instance is fairly expensive,
//  and you rarely need access to multiple instances within a single process.
    private static AppDatabase mAppDatabase;

    public static ApplicationController getInstance() {
        return mInstance;
    }

    @Override
    public void onCreate() {
        super.onCreate();

         mInstance = this;

         // Get a database instance to work with
         mAppDatabase = Room.databaseBuilder(getApplicationContext(),
                 AppDatabase.class, Constants.DB_NAME).build();
    }

    public static AppDatabase getAppDatabase(){
        return mAppDatabase;
    }
}
```

7. Connect your application class with your manifest by adding application class as the name of **<application>** tag:

```
<application
        android:name=".ApplicationController"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

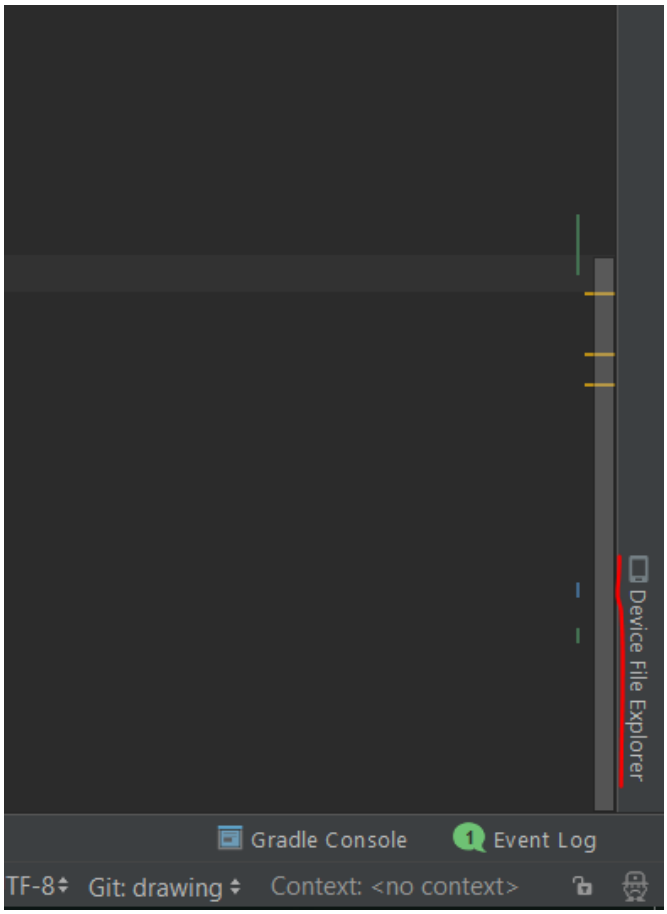
```
        <activity android:name=".activities.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".activities.Main2Activity"></activity>
    </application>
```

8. Create the repository class:

```
  --> the best practice
// A Repository mediates between the domain and data mapping layers,
// acting like an in-memory domain object collection.
// We access the database class and the DAO class from the repository and perform list of operations
// such as insert, update, delete, get etc.
public class UserRepository {
    private AppDatabase appDatabase;

    public UserRepository(Context context) {
        appDatabase = ApplicationController.getAppDatabase();
    }

    public void insertTask(final User user,
                           final OnUserRepositoryActionListener listener)   {
        new InsertTask(listener).execute(user);
    }

    public User getUserByName(String firstName, String lastName){
        return appDatabase.userDao().findByName(firstName, lastName);
    }

// DO NOT PERFORM OPERATION ON MAIN THREAD AS APP WILL CRASH
   // See more details about AsyncTask in the next chapter
   private class InsertTask extends AsyncTask<User, Void, Void> {
       OnUserRepositoryActionListener listener;
        InsertTask(OnUserRepositoryActionListener listener) {
            this.listener = listener;
        }
        @Override
        protected Void doInBackground(User... users) {
            appDatabase.userDao().insertTask(users[0]);
```

```
            return null;
        }
        @Override
        protected void onPostExecute(Void aVoid) {
            super.onPostExecute(aVoid);
            listener.actionSuccess();
        }
    }
}

// You can use an interface like this to perform actions on main thread
// when the action is done.
public interface OnUserRepositoryActionListener {
    void actionSuccess();
    void actionFaailed();
}
```

Access database content



Then go to `/data/data/{package_name}/databases/`

Right-click and save your database
Download SQLite Browser from here:
https://sqlitebrowser.org/dl/
Open your database file

AsyncTask

Android **AsyncTask** is an abstract class provided by Android which gives us the liberty to perform heavy tasks in the background and keep the UI thread light thus making the application more responsive.

We use android AsyncTask to perform the heavy tasks in background on a dedicated thread and passing the results back to the UI thread. Hence use of AsyncTask in android application keeps the UI thread responsive at all times.

The basic methods used in an android AsyncTask class are defined below :

- **doInBackground()** : This method contains the code which needs to be executed in background. In this method we can send results multiple times to the UI thread by publishProgress() method. To notify that the background processing has been completed we just need to use the return statements

- **onPreExecute()** : This method contains the code which is executed before the background processing starts

- **onPostExecute()** : This method is called after doInBackground method completes processing. Result from doInBackground is passed to this method

- **onProgressUpdate()** : This method receives progress updates from doInBackground method, which is published via publishProgress method, and this method can use this progress update to update the UI thread

The three generic types used in an android AsyncTask class are given below :

- **Params** : The type of the parameters sent to the task upon execution
- **Progress** : The type of the progress units published during the background computation
- **Result** : The type of the result of the background computation

!!! The AsyncTask instance must be created and invoked in the UI thread.
!!! The methods overridden in the AsyncTask class should never be called. They're called automatically.
!!! AsyncTask can be called only once. Executing it again will throw an exception

Example:

```
private class AsyncTaskRunner extends AsyncTask<String, String, String> {

    private String resp;
    ProgressDialog progressDialog;

    @Override
    protected String doInBackground(String... params) {
        publishProgress("Sleeping..."); // Calls onProgressUpdate()
        try {
            int time = Integer.parseInt(params[0])*1000;
            Thread.sleep(time);
            resp = "Slept for " + params[0] + " seconds";
        } catch (InterruptedException e) {
            e.printStackTrace();
            resp = e.getMessage();
        } catch (Exception e) {
            e.printStackTrace();
            resp = e.getMessage();
        }
        return resp;
    }

  @Override
    protected void onPostExecute(String result) {
        // execution of result of Long time consuming operation
        progressDialog.dismiss();
        finalResult.setText(result);
```

```
    }


    @Override
    protected void onPreExecute() {
        progressDialog = ProgressDialog.show(MainActivity.this,
                "ProgressDialog",
                "Wait for "+time.getText().toString()+ " seconds");
    }
```

JSON

The following JSON example defines an employees object, with an array of 3 employee records:

JSON Example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

The following XML example also defines an employees object with 3 employee records:

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation.

It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

http://www.zoftino.com/handling-json-using-gson-in-android