Lab on apps development for tablets, smartphones and smartwatches

Week 5: Data Management

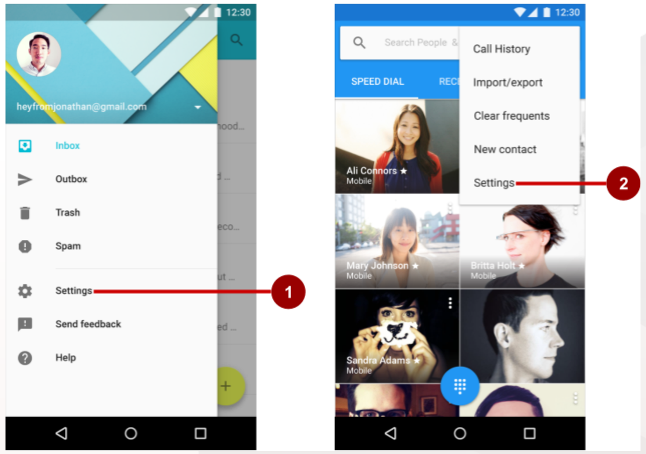
Dr. Marina Zapater, Prof. David Atienza

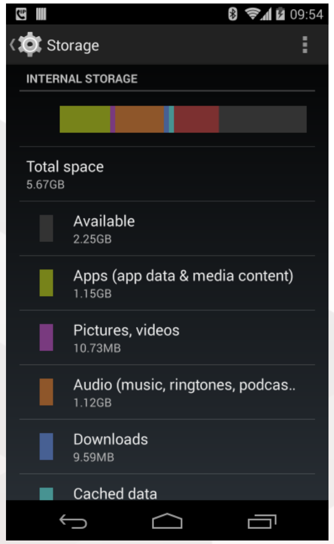
Mr. Grégoire Surrel, Ms. Elisabetta de Giovanni,
Mr. Dionisijie Sopic, Ms. Halima Najibi, Ms. Farnaz Forooghifar

Embedded Systems Laboratory (ESL) – Faculty of Engineering (STI)

Outline of the class

- **Shared Preferences** *(Lab 6)*
  - Creating/saving/restoring prefs.
  - Setting UI
- Internal/External storage
  - Writing to files
- SQLite *(Lab7)*
- Firebase *(Today's lab 5!)*




Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

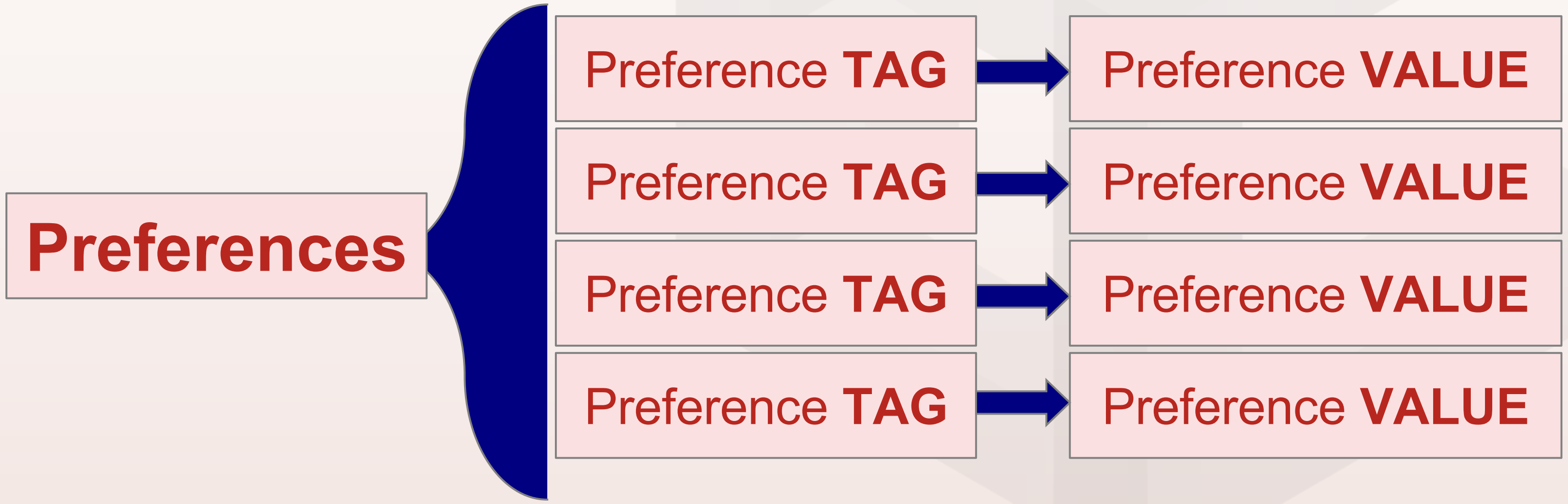
Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

Cached data

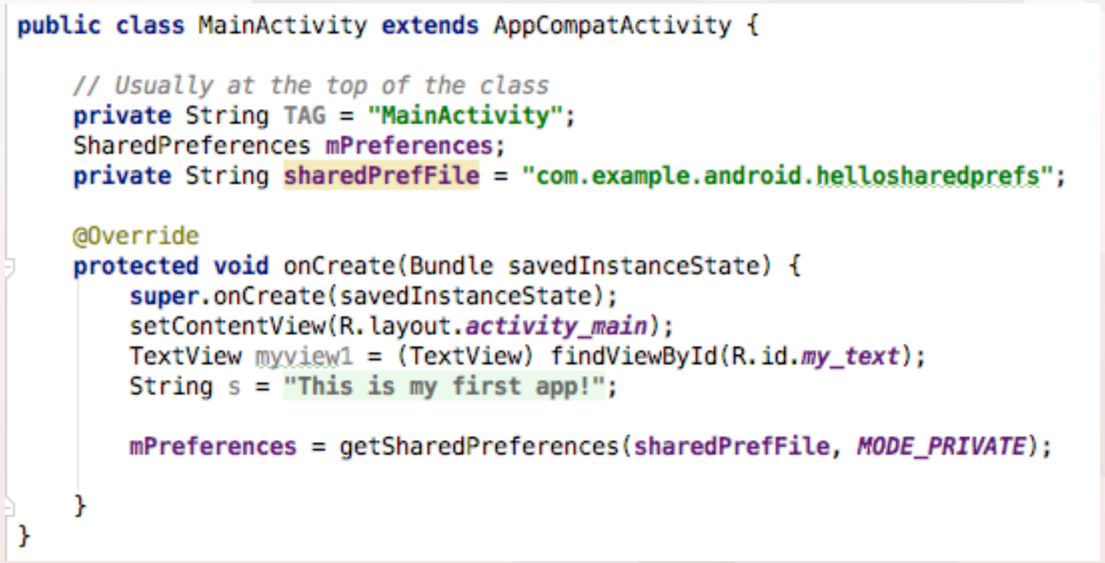
Shared Preferences

- Preferences are a convenient way to store config parameters
- Read/write small amounts of data as key/value pairs
- “Preferences” could be either private or shared
  - Shared preferences: other applications could potentially read them
  - Private means that they could be restricted at:
    - Application level
    - Activity level



Creating Shared Preferences

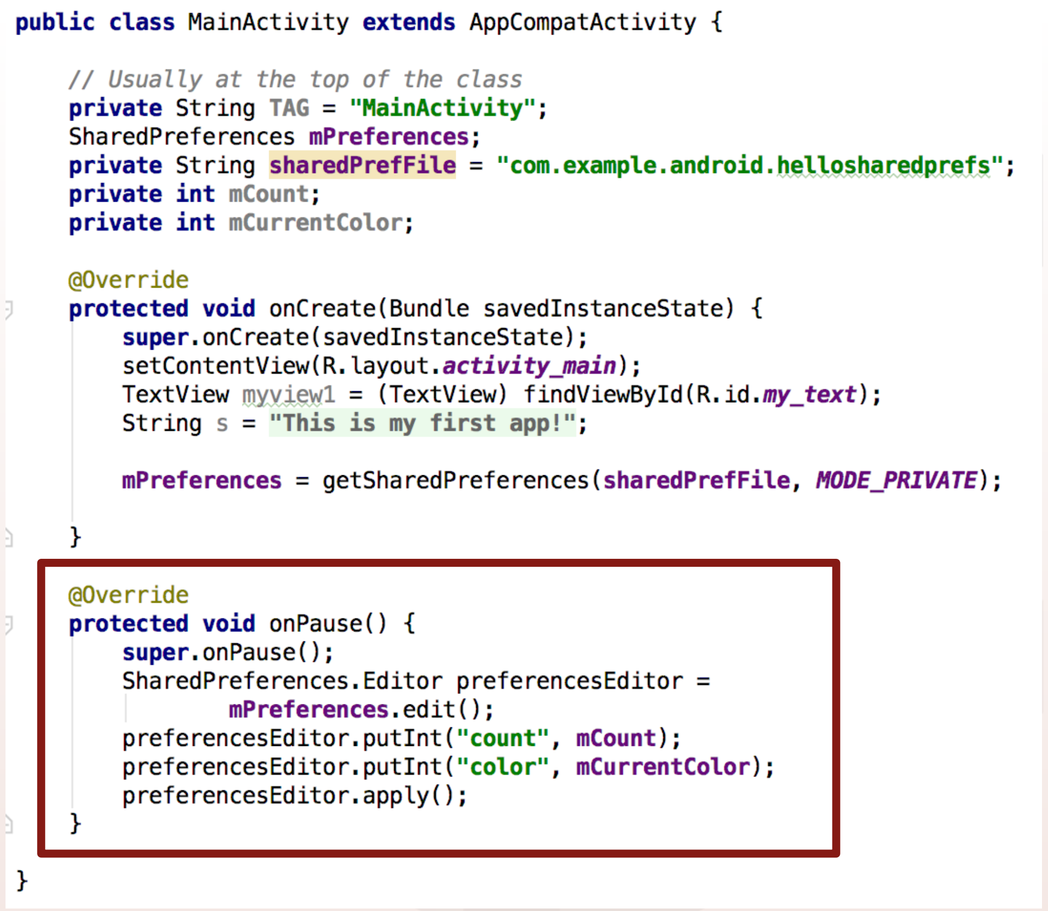
- We need only one Share Preferences file per app.
- Name it with the package name of your app
  - Unique and easy way to associate it with an app.
  - MODE argument for getSharedPreferences() is for backwards compatibility—use only MODE_PRIVATE to be secure



```
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

    }
}
```

Saving preferences

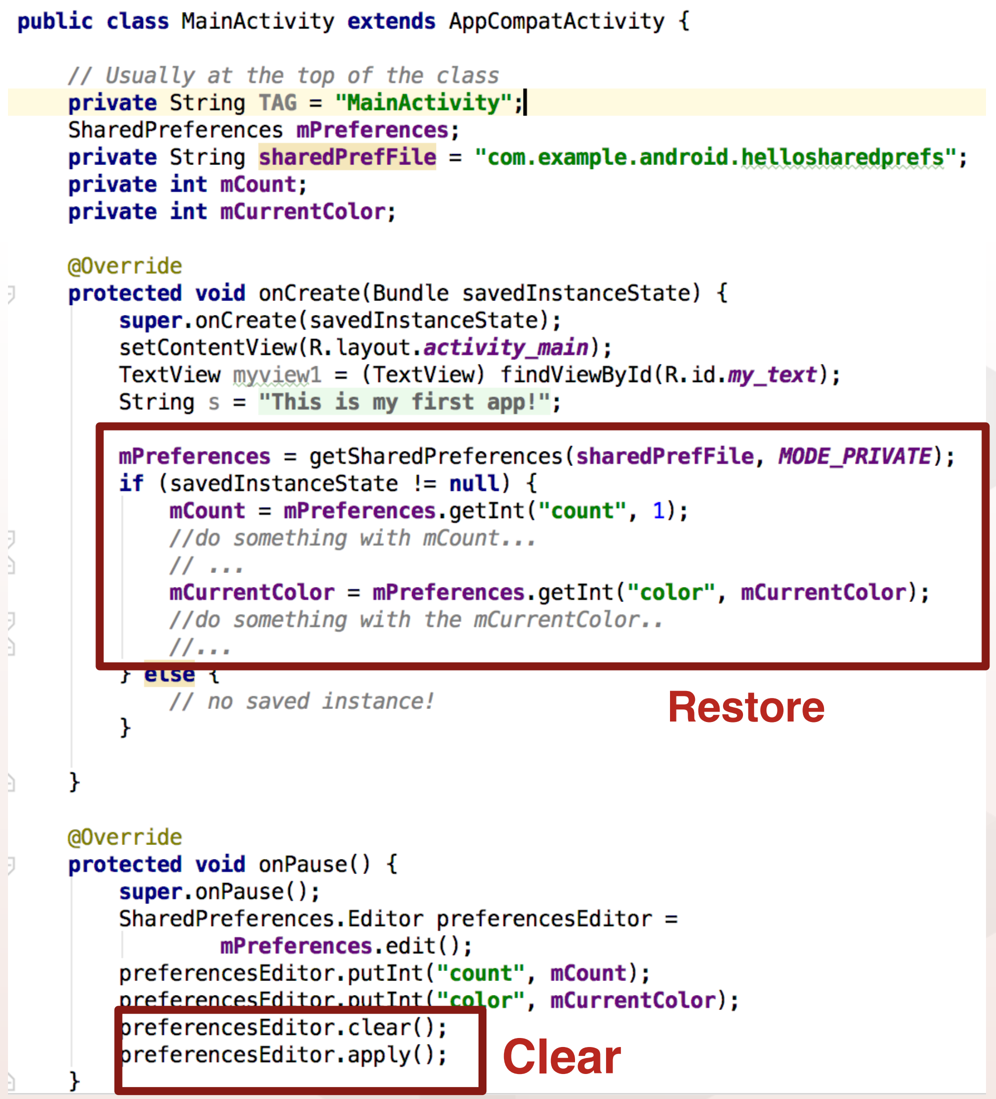
- How to edit preferences? → using SharedPreferences.Editor
  - This takes care of all file operations
  - Careful! Overwrite in case the key already exists
- Be sure to commit operations at the end:
  - apply() saves asynchronously and safely



```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";
    private int mCount;
    private int mCurrentColor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

    }

    @Override
    protected void onPause() {
        super.onPause();
        SharedPreferences.Editor preferencesEditor =
                mPreferences.edit();
        preferencesEditor.putInt("count", mCount);
        preferencesEditor.putInt("color", mCurrentColor);
        preferencesEditor.apply();
    }
}
```

Restoring and clearing preferences

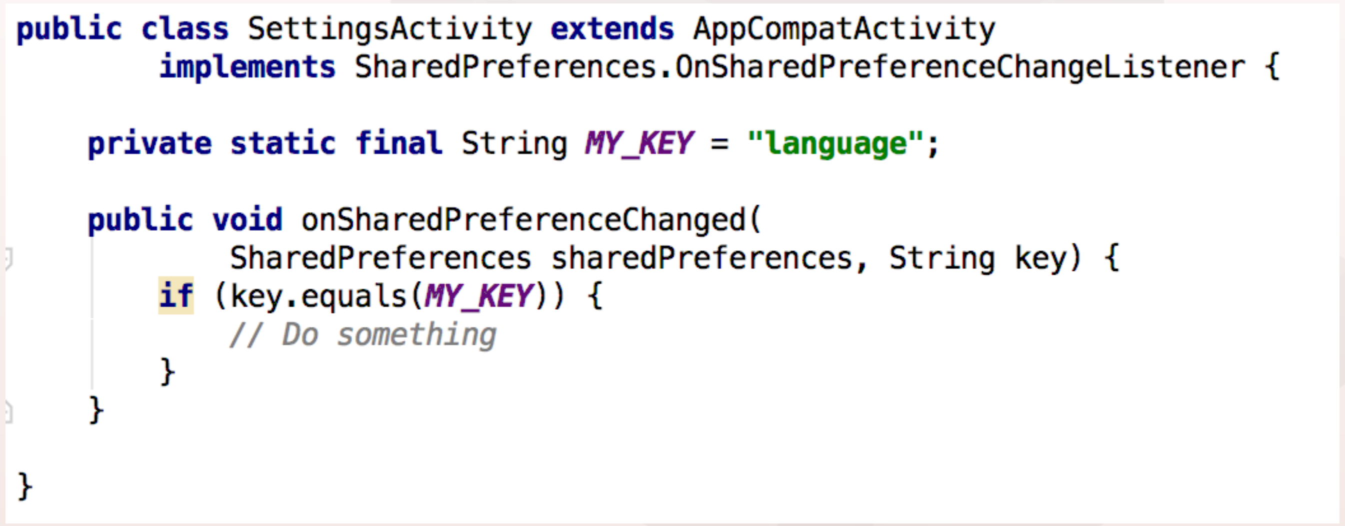
- Restore in onCreate() in Activity
  - Get methods take two arguments:
    - the key
    - the default value if the key cannot be found
  - Use default argument so you do not have to test whether the preference exists in the file
- Clearing:
  - Call clear() on the SharedPreferences.Editor and apply changes



```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";
    private int mCount;
    private int mCurrentColor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
        if (savedInstanceState != null) {
            mCount = mPreferences.getInt("count", 1);
            //do something with mCount...
            // ...
            mCurrentColor = mPreferences.getInt("color", mCurrentColor);
            //do something with the mCurrentColor..
            //...
        } else {
            // no saved instance!
        }

    }

    @Override
    protected void onPause() {
        super.onPause();
        SharedPreferences.Editor preferencesEditor =
                mPreferences.edit();
        preferencesEditor.putInt("count", mCount);
        preferencesEditor.putInt("color", mCurrentColor);
        preferencesEditor.clear();
        preferencesEditor.apply();
    }
```

Settings Activity

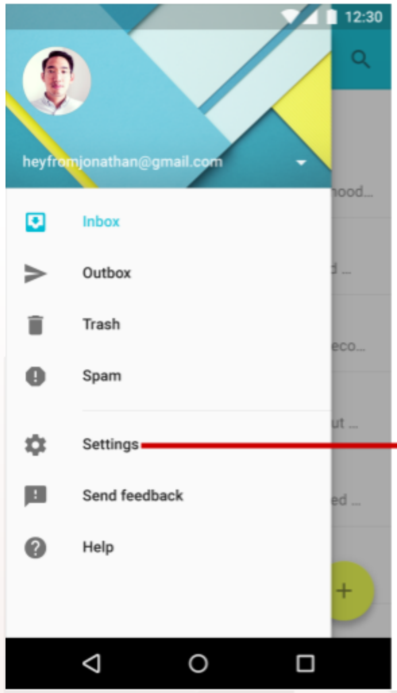
- Creating a “SettingsActivity” that we can have in a menu of our app
- Listening to changes on the application settings
  - Implement interface SharedPreference.OnSharedPreferenceChangeListener
  - Register listener with registerOnSharedPreferenceChangeListener()
  - Register and unregister listener in onResume() and onPause()
  - Implement on onSharedPreferenceChanged() callback

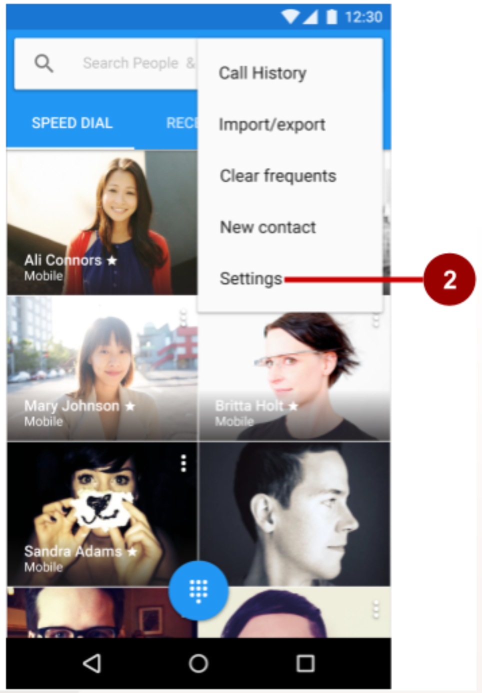


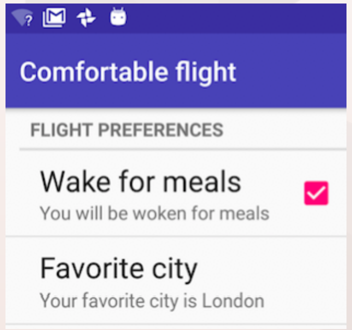
```java
public class SettingsActivity extends AppCompatActivity
        implements SharedPreferences.OnSharedPreferenceChangeListener {

    private static final String MY_KEY = "language";

    public void onSharedPreferenceChanged(
            SharedPreferences sharedPreferences, String key) {
        if (key.equals(MY_KEY)) {
            // Do something
        }
    }

}
```

Settings User Interface

- How about the user interface?
  - How do we create the "Settings UI"?
- Two ways to access settings:
  1. Navigation drawer
  2. Options menu
- Use Preference object instead of View objects in your settings screen
- Design and edit Preference objects in the Layout editor just like you do for View objects
  - CheckBoxPreference, EditTextPreference, SwitchPreference



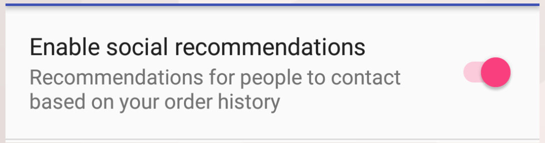



Comfortable flight

FLIGHT PREFERENCES

Wake for meals
You will be woken for meals

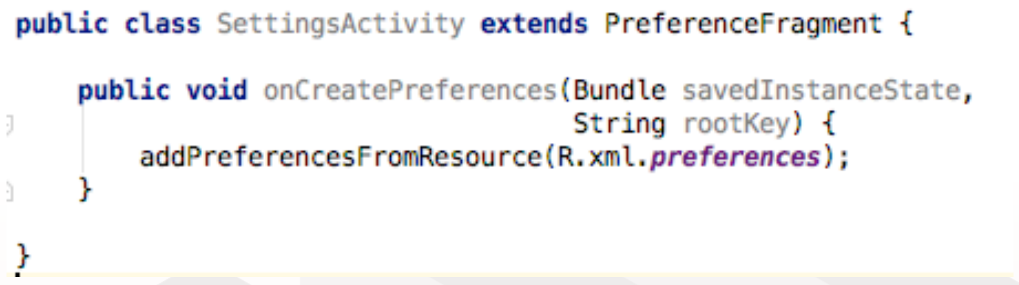
Favorite city
Your favorite city is London


Enable social recommendations
Recommendations for people to contact based on your order history

Activities and Fragments for settings

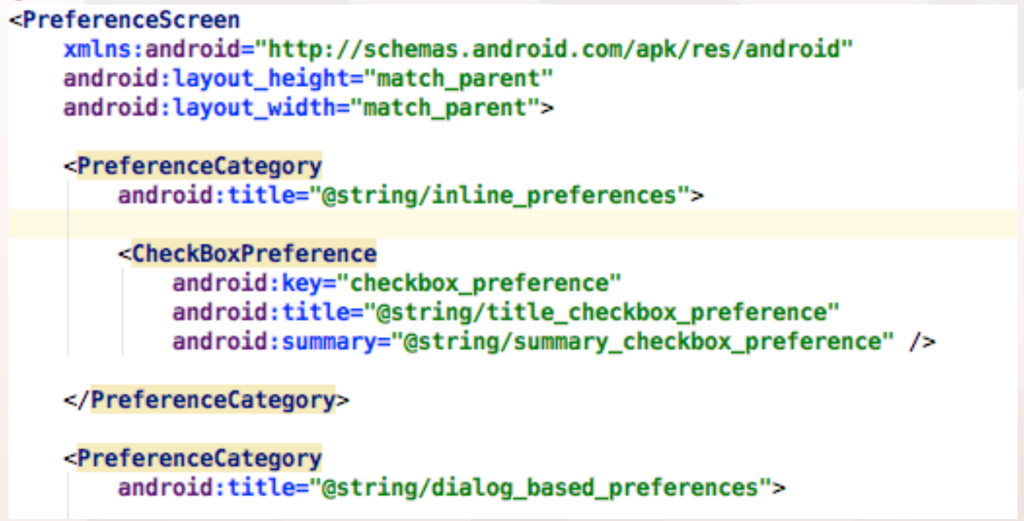
- AppCompatActivity with PreferenceFragment
  - Android 3.0 and newer
- Link preference activity to the XML file → R.xml.preferences
- Set preference theme in styles.xml

SettingsActivity



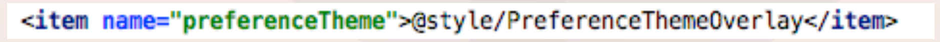
```
public class SettingsActivity extends PreferenceFragment {

    public void onCreatePreferences(Bundle savedInstanceState,
                                    String rootKey) {
        addPreferencesFromResource(R.xml.preferences);
    }

}
```

preferences.xml



```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <PreferenceCategory
        android:title="@string/inline_preferences">

        <CheckBoxPreference
            android:key="checkbox_preference"
            android:title="@string/title_checkbox_preference"
            android:summary="@string/summary_checkbox_preference" />

    </PreferenceCategory>

    <PreferenceCategory
        android:title="@string/dialog_based_preferences">
```

Styles.xml

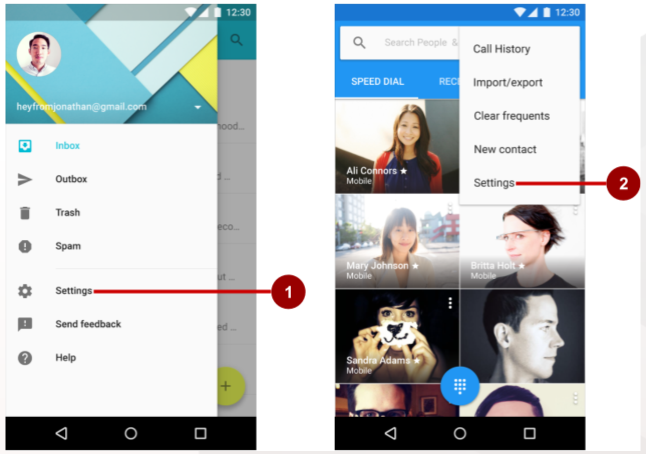


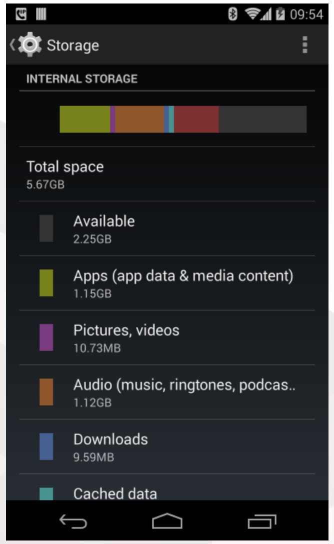
```
<item name="preferenceTheme">@style/PreferenceThemeOverlay</item>
```

We will exercise this in Week 6!!

Outline of the class

- Shared Preferences
- **Internal/External storage**
  - Writing to files
- SQLite
- Firebase




Storage

INTERNAL STORAGE

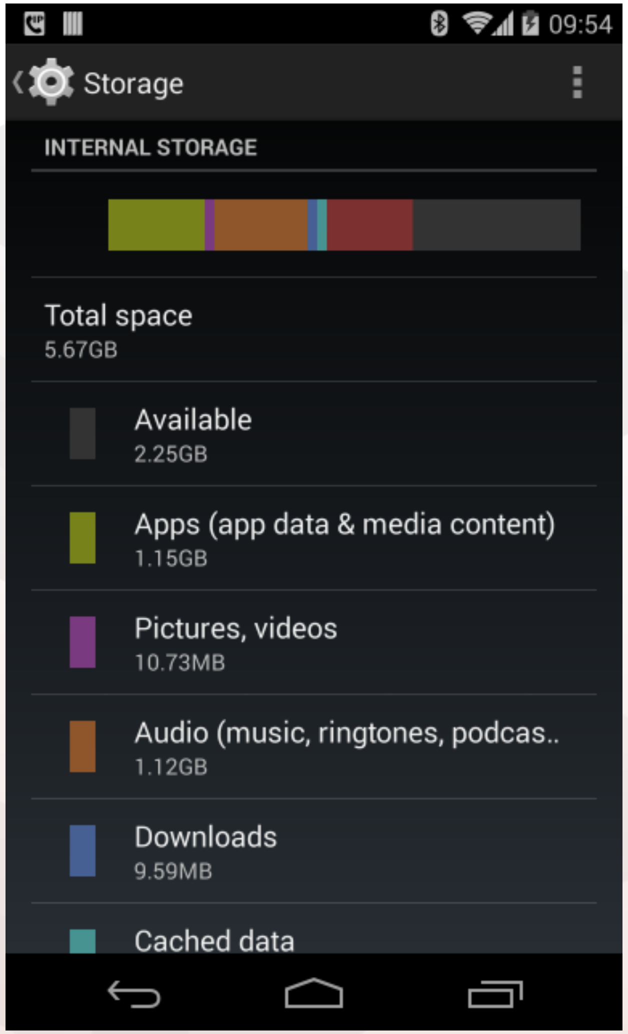
Total space 5.67GB
Available 2.25GB
Apps (app data & media content) 1.15GB
Pictures, videos 10.73MB
Audio (music, ringtones, podcas.. 1.12GB
Downloads 9.59MB
Cached data

The Android FileSystem

- Internal storage -- Private directories just for your app
  - Always available
  - Uses device's filesystem
  - Only your app can access files, unless explicitly set to be readable or writable
  - On app uninstall, system removes all app's files from internal storage
- External storage -- Public directories
  - Not always available, can be removed
  - Uses device's file system or physically external storage like SD card
  - World-readable, so any app can read
  - On uninstall, system does not remove files private to app

Internal storage

- Uses private directories just for your app
- App always has permission to read/write
  - **Where?** /data/data/<package>/files
  - **How?** Use standard java I/O classes
    - Permanent storage directory—getFilesDir()
    - Temporary storage directory—getCacheDir()


09:54

Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

Cached data

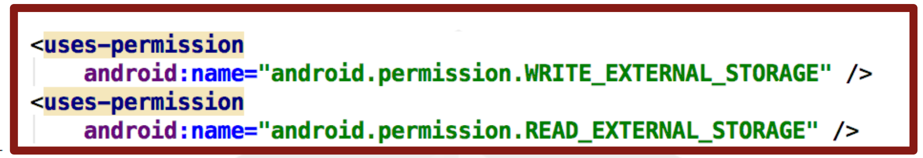
External storage

- On device or SD card
  - **Where?** Environment.getExternalStorageDirectory()
  - **How?** Use standard java I/O classes

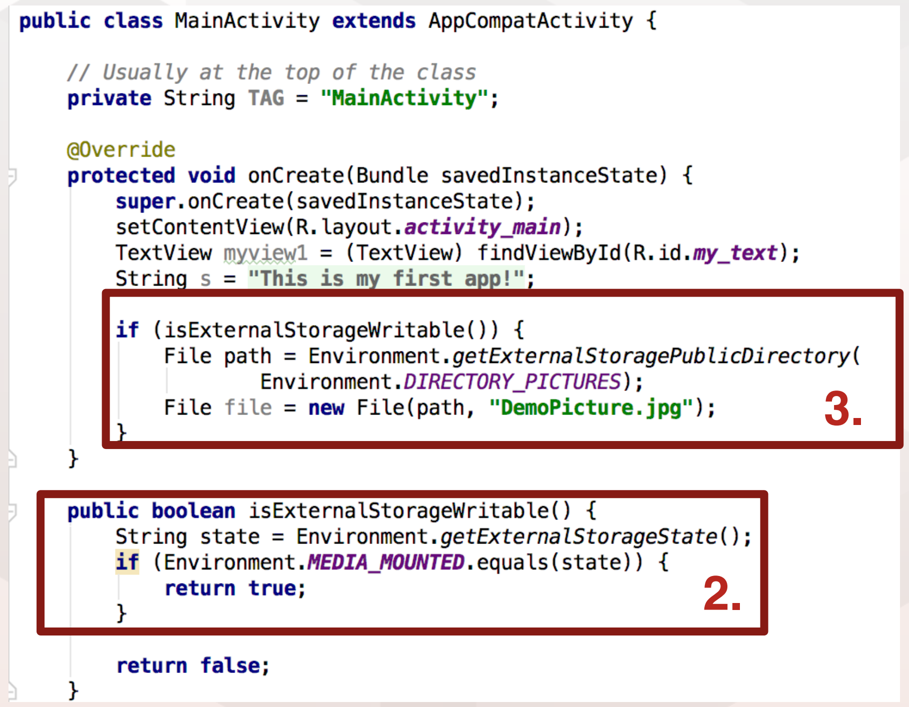
1. Set permissions in Android Manifest
   - Write permission includes read permission
2. Always check availability of storage!
3. Get the path to storage folder and create a file

- When you no longer want the file, you can delete it.

1.



```xml
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```



```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        if (isExternalStorageWritable()) {
            File path = Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_PICTURES);
            File file = new File(path, "DemoPicture.jpg");
        }
    }

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

        return false;
    }
```

3.

2.

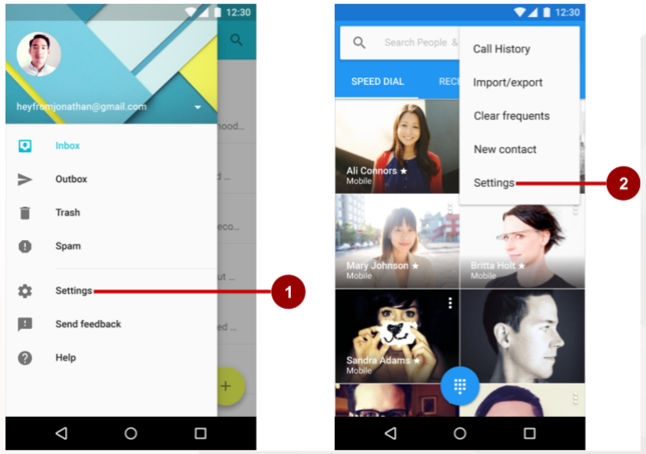
Important notes about external storage!

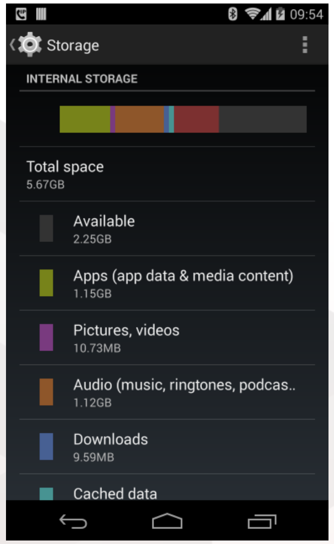
- What happens if there is not enough space?
  - If there is not enough space, throws IOException
  - If you know the size of the file, check against space
    - getFreeSpace()
    - getTotalSpace().
  - If you do not know how much space is needed
    - try/catch IOException

- Do not delete the user’s files!!
  - When the user uninstalls your app, your app's private storage directory and all its contents are deleted
  - ***Do not use private storage for content that belongs to the user!***

Outline of the class

- Shared Preferences
- Internal/External storage
  - Writing to files
- **SQLite** → Room library
- Firebase




Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

Cached data

SQLite

- A database to store structured information
  - Store data in tables of rows and columns (spreadsheet…)
  - Field = intersection of a row and column
  - Rows are identified by unique IDs
  - Column names are unique per table
- General purpose solution:
  - Lightweight database based on SQL
  - Ideal for repeating or structured data, such as contacts
- Android provides SQL-like database with standard SQL syntax

```
SELECT name FROM table WHERE name = “Luca”
```

Using the "Room library" to make our lives easier

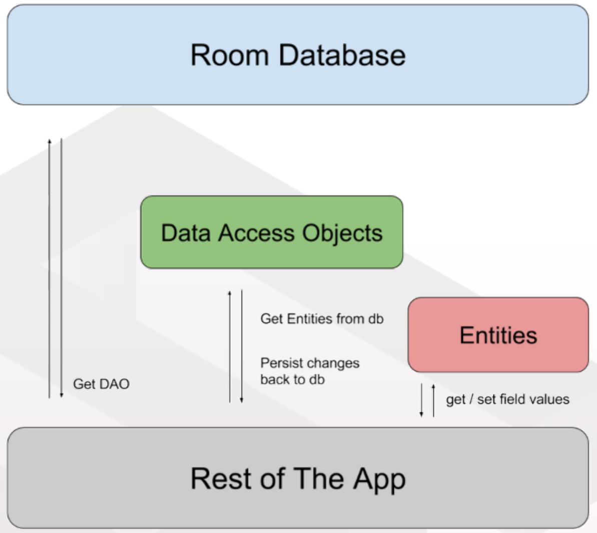
- Room provides an abstraction layer over SQLite
- Three major components (.java files):
  - Database: main access point to DB

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

  - Entity: represents a table within the database
  - Data Access Objects (DAO): Methods used for accessing the database

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

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

    @ColumnInfo(name = "last_name")
    private String lastName;

    // Getters and setters are ignored for brevity,
    // but they're required for Room to work.
}
```

```
@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);
}
```



We will exercise this in Week 7!!

Using the room database

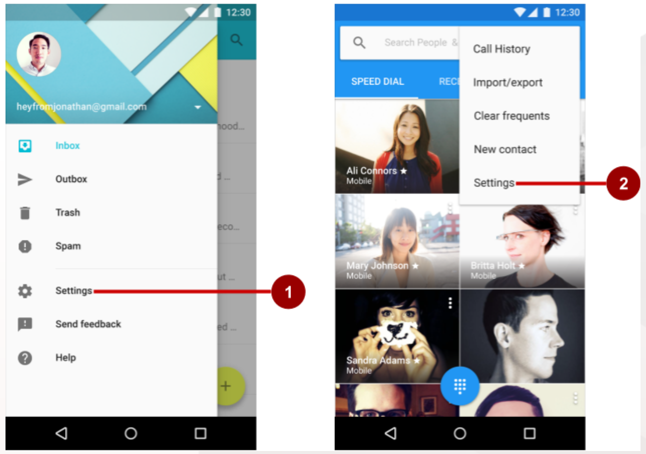
- To use a database, you get an instance of it using the following code:

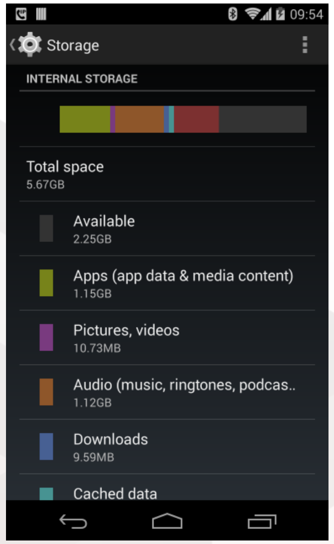
```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "database-name").build();
```

- Queries to the database need to be filled into the DAO:
  - Insert rows:
    - INSERT INTO table (field1,... fieldN) VALUES (value1,..., valueN);
  - Delete rows:
    - DELETE FROM table WHERE column=“value”
  - Update rows:
    - UPDATE table SET column=“value” WHERE condition;
  - Retrieve rows that meet given criteria
    - SELECT columns FROM table WHERE column="value"

Outline of the class

- Shared Preferences
- Internal/External storage
  - Writing to files
- SQLite
- **Firebase**

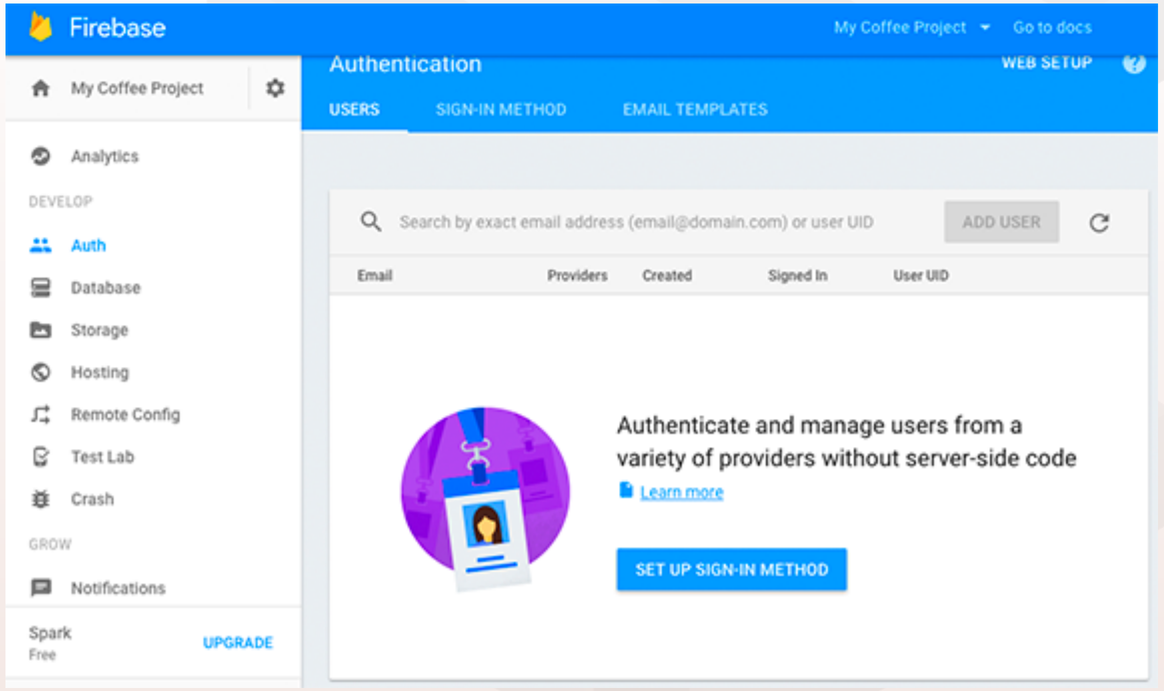



09:54
Storage
INTERNAL STORAGE
Total space
5.67GB
Available
2.25GB
Apps (app data & media content)
1.15GB
Pictures, videos
10.73MB
Audio (music, ringtones, podcas..
1.12GB
Downloads
9.59MB
Cached data

What is Firebase?

- Firebase is a platform that provides tools to help you
  - develop your app
  - grow your user base
  - earn money from your app

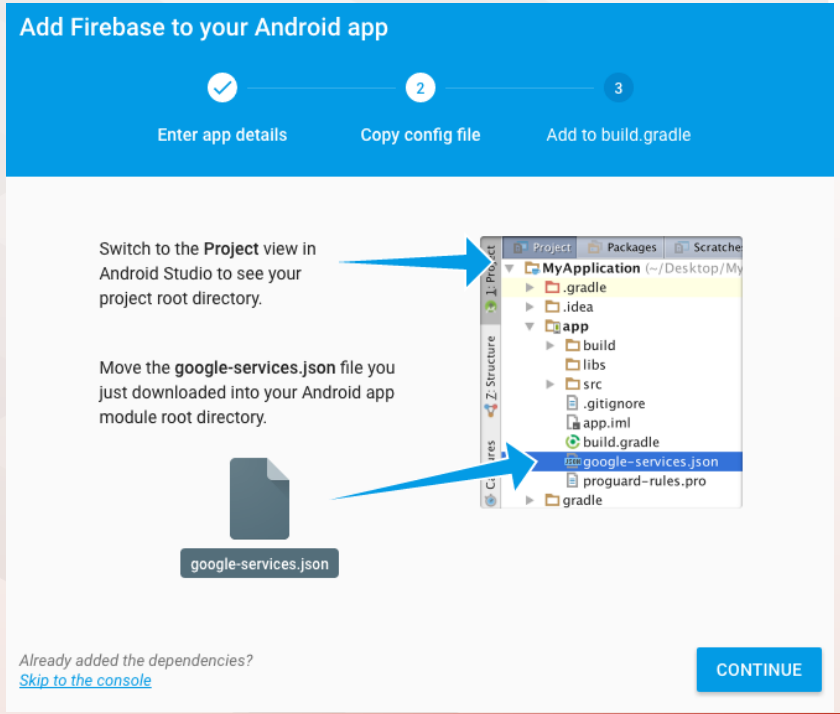
- We will show you how to use it to sync data to the cloud
  1. Connect your app to your Firebase project
  2. Enable Firebase features in the console
  3. Add code to your app (where needed)



Authenticate and manage users from a variety of providers without server-side code

Firebase console

- Going to firebase.google.com
  - The console allows you to create new projects
  - Firebase creates a config file for your app
    - You can add the config file to your project
- All this happens automatically through Android Studio!
  - We will teach you how to do it, step by step, during the lab.


Add Firebase to your Android app

Switch to the **Project** view in Android Studio to see your project root directory.

Move the **google-services.json** file you just downloaded into your Android app module root directory.

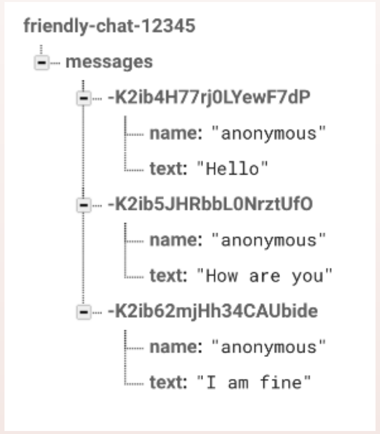
Already added the dependencies?
Skip to the console

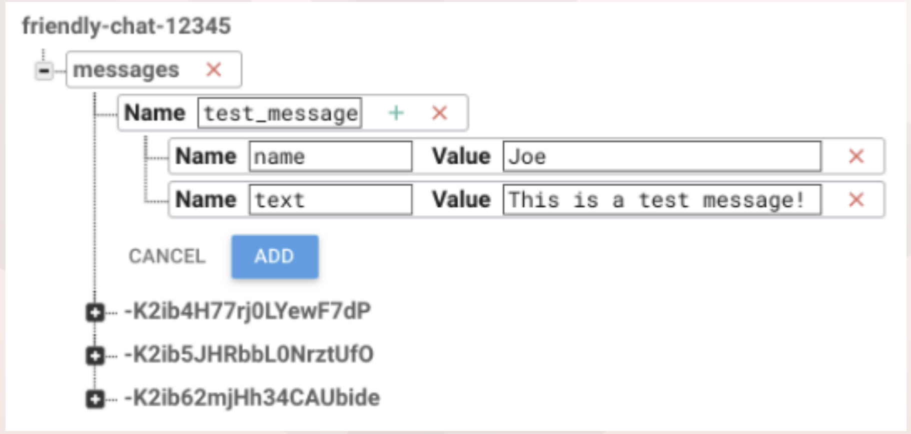
Using Firebase al realtime Database

- We will use Firebase as a Database
  - Check the web! It has many other features!
- Data is synced across all clients, and remains available when your app goes offline
  - Data is stored as JSON
  - Firebase is in fact a NoSQL database
    - Key-value database
    - Internally different from SQL database

Database

Store and sync data in realtime across all connected clients


```
friendly-chat-12345
  messages
    -K2ib4H77rj0LYewF7dP
      name: "anonymous"
      text: "Hello"
    -K2ib5JHRbbL0NrztUfO
      name: "anonymous"
      text: "How are you"
    -K2ib62mjHh34CAUbide
      name: "anonymous"
      text: "I am fine"
```


```
friendly-chat-12345
  messages ✕
    Name test_message + ✕
      Name name   Value Joe ✕
      Name text   Value This is a test message! ✕
    CANCEL  ADD
    -K2ib4H77rj0LYewF7dP
    -K2ib5JHRbbL0NrztUfO
    -K2ib62mjHh34CAUbide
```

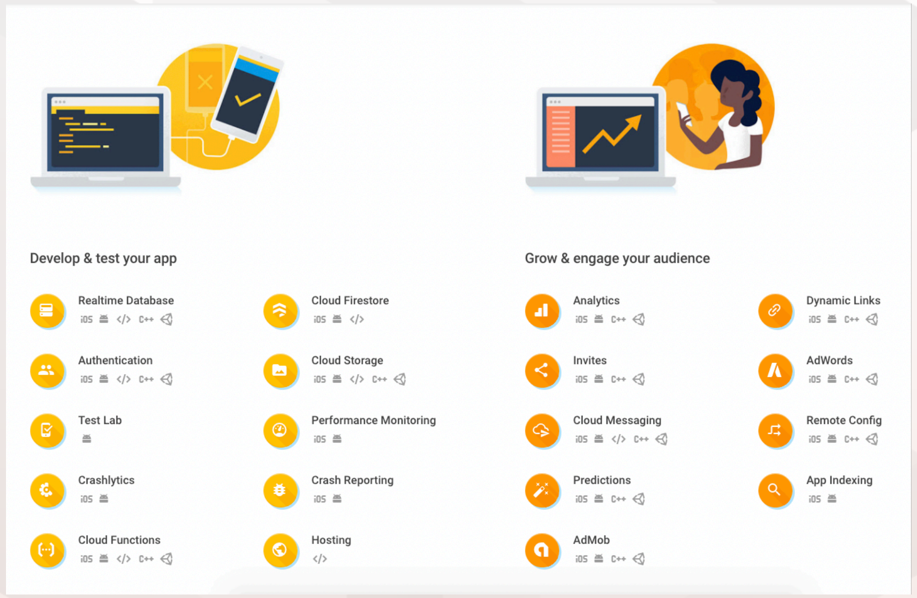
How to use Firebase database in your app?

- In your app project, add dependency in app/build.gradle: compile 'com.google.firebase:firebase-database:n.n.n'
- In your app source code, put data in the database, and get data from the database (API Reference)
  - FirebaseDatabase database = FirebaseDatabase.getInstance();
  - DatabaseReference myRef = database.getReference(*path*);
  - myRef.setValue("New value");
- All details in the Lab session!

Today's lab

- Exercising Firebase!
  - Connecting the Profile app to Firebase
  - Adding real-time database to your app
  - Configure Firebase Database rules

- Writing/reading data from Firebase storage


Develop & test your app

Realtime Database
Cloud Firestore
Authentication
Cloud Storage
Test Lab
Performance Monitoring
Crashlytics
Crash Reporting
Cloud Functions
Hosting

Grow & engage your audience

Analytics
Dynamic Links
Invites
AdWords
Cloud Messaging
Remote Config
Predictions
App Indexing
AdMob

Questions?