Android Development

Lecture 6
Data Persistence

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Lecture Summary

- Shared Preferences
- Internal Storage
- External Storage
- JSON Appendix
- SQLite Database

Android & Data Persistence

Shared Preferences

Store private primitive data in a key-value structure.

Internal Storage & External Storage

Store data on device memory (private data) or on external storage (public data).

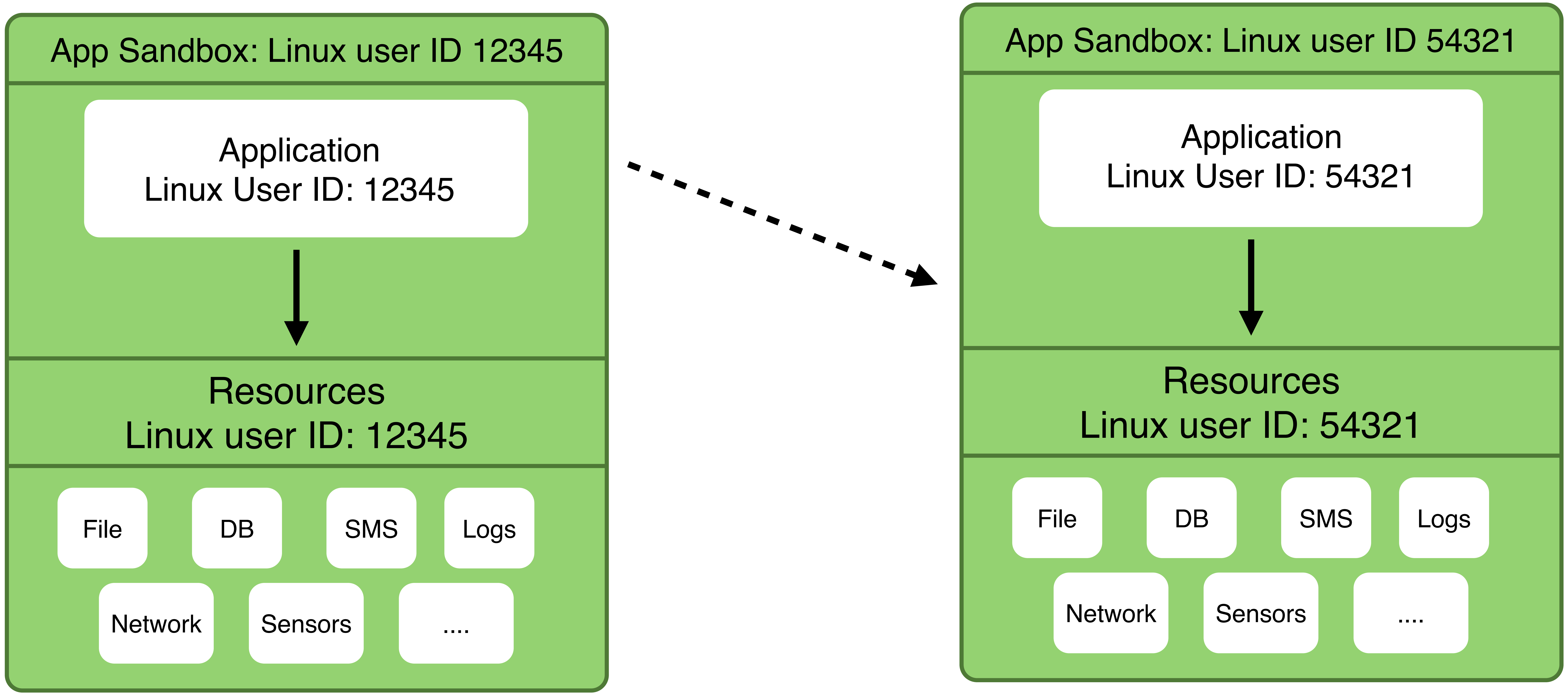
SQLite Database

Store structured data in a private database.

Network Connection

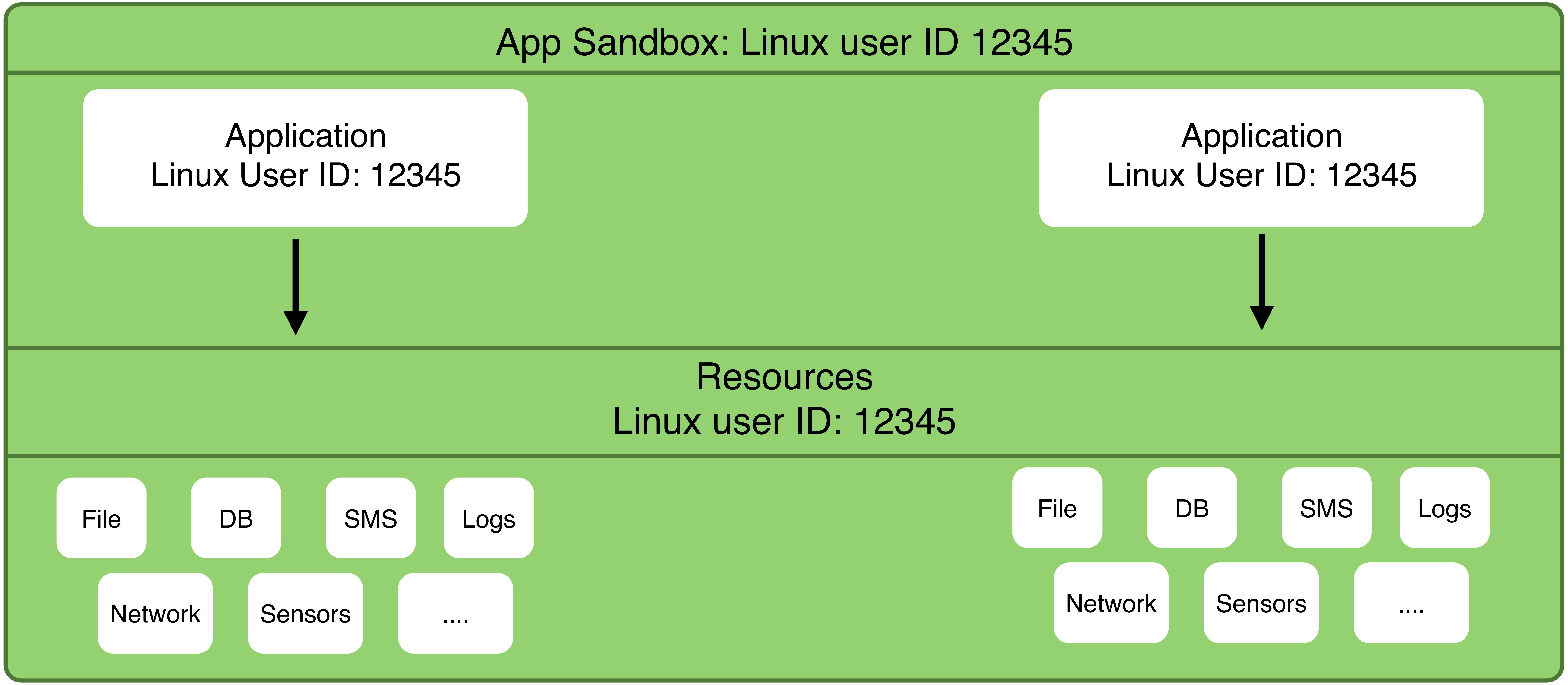
Store data on a remote server or service.

Application Sandbox



Source: PA-Chapter 3 - http://www.ibm.com/developerworks/xml/library/x-androidsecurity/ - http://source.android.com/tech/security/index.html#the-application-sandbox

Application Sandbox



Source: PA-Chapter 3 - http://www.ibm.com/developerworks/xml/library/x-androidsecurity/ - http://source.android.com/tech/security/index.html#the-application-sandbox

Shared Preferences

- May applications need a lightweight data storage mechanism for storing **application state**, **configuration options**, **simple user information**, and other **user's data**.
- This mechanism on Android platform is called "Shared Preferences" and provides a simple preferences system for storing primitive application data at the Activity level.
- The preferences are not shared across all of application's activities and it is not possible to share preferences outside of the package.
- Supported data types are:
  - Boolean values
  - Float values
  - Integer values
  - Long values
  - String values

Shared Preferences

- The **SharedPreferences** class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types.
- To get a SharedPreferences object for your application, use one of two methods:
  - **getSharedPreferences()** - Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
  - **getPreferences()** - Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.
- To write values:
  - Call edit() to get a SharedPreferences.Editor.
  - Add values with methods such as putBoolean() and putString().
  - Commit the new values with commit()
- To read values, use SharedPreferences methods such as getBoolean() and getString().

Shared Preferences

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
    }

    @Override
    protected void onStop(){
       super.onStop();

      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);
      // Commit the edits!
      editor.commit();
    }
}
```

Read Shared Preferences

Other get methods are: getFloat(...), getInt(...), getLong(...), getString(...)

default value

Other put methods are: putFloat(...), putInt(...), putLong(...), putString(...)

Commit all changes from this editing session.

Edit/Add Shared Preferences

http://developer.android.com/guide/topics/data/data-storage.html

Internal Storage / Application Directories

- Data of Android application is stored on the Android file system in the following directory:
  - /data/data/<package_name>/
- In particular the API allows to get references to list, read and write files from the following subdirectories:
  - /data/data/<package_name>/files
  - /data/data/<package_name>/cache
- Several default subdirectories are created for storing databases, preferences, and files as necessary. You can also create your custom directory as needed.
- The Context object is used to perform file operations and interact with the file system for application file management.

Internal Storage / Application Directories

- You can save files directly on the device's internal storage.
- By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.
- To create and write a private file to the internal storage:
  - Call openFileOutput() with the name of the file and the operating mode. This returns a FileOutputStream.
  - Write to the file with write().
  - Close the stream with close().

Write to Internal Storage

```
String filename = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(filename, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

- MODE_PRIVATE will create the file (or replace a file of the same name) and make it private to your application. Other modes available are: MODE_APPEND, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE.
- To read a file from internal storage:
- Call openFileInput() and pass it the name of the file to read. This returns a FileInputStream.
- Read bytes from the file with read().
- Then close the stream with close().

Read from Internal Storage

```
String filename = "hello_file";

FileInputStream fis = openFileInput(filename);
StringBuffer sBuffer = new StringBuffer();
DataInputStream dataIO = new DataInputStream(fis);

String strLine = null;

while((strLine = dataIO.readLine()) != null)
{
  sBuffer.append(strLine+"\n");
}

dataIO.close();
fis.close();
```

- There is a shortcut for reading files stored in the default /files subdirectory.
- You can use the openFileInput method to obtain a FileOutputStream reference and read the file content using standard Java API.
- For example you can use a DataInputStream object for reading a file, line by line, and store it in a StringBuffer.

External Storage

- Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.
- **Caution**: External files can disappear if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.
- Before you do any work with the external storage, you should always call getExternalStorageState() to check whether the media is available. The media might be mounted to a computer, missing, read-only, or in some other state.

External Storage

- The getExternalStorageState() method allow to check if the external memory is available to read and write and also returns other states that you might want to check, such as whether the media is being shared (connected to a computer), is missing entirely, has been removed badly, etc. You can use these to notify the user with more information when your application needs to access the media.

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    //  to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

External Storage

- If you're using API Level 8 or greater, use getExternalFilesDir() to open a File that represents the external storage directory where you should save your files.
- This method takes a type parameter that specifies the type of subdirectory you want, such as DIRECTORY_MUSIC and DIRECTORY_RINGTONES (pass null to receive the root of your application's file directory). This method will create the appropriate directory if necessary. By specifying the type of directory, you ensure that the Android's media scanner will properly categorize your files in the system (for example, ringtones are identified as ringtones and not music). If the user uninstalls your application, this directory and all its contents will be deleted.
- If you're using API Level 7 or lower, use getExternalStorageDirectory(), to open a File representing the root of the external storage. You should then write your data in the following directory:
  - /Android/data/<package_name>/files/
  - The <package_name> is your Java-style package name, such as "com.example.android.app". If the user's device is running API Level 8 or greater and they uninstall your application, this directory and all its contents will be deleted.

External Storage

```
File dir = Environment.getExternalStorageDirectory();
String path = dir.getAbsolutePath();
String fileName = "mobdev_bookmarklist.txt";

File outputFile = new File(path + File.separator + fileName);

if(!outputFile.exists())
   outputFile.createNewFile();

FileOutputStream fos = new FileOutputStream(outputFile);
fos.write(...);

fos.close();
```

- In order to write files to the external storage an application needs to declare a specific permission called WRITE_EXTERNAL_STORAGE in the ApplicationManifest.

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

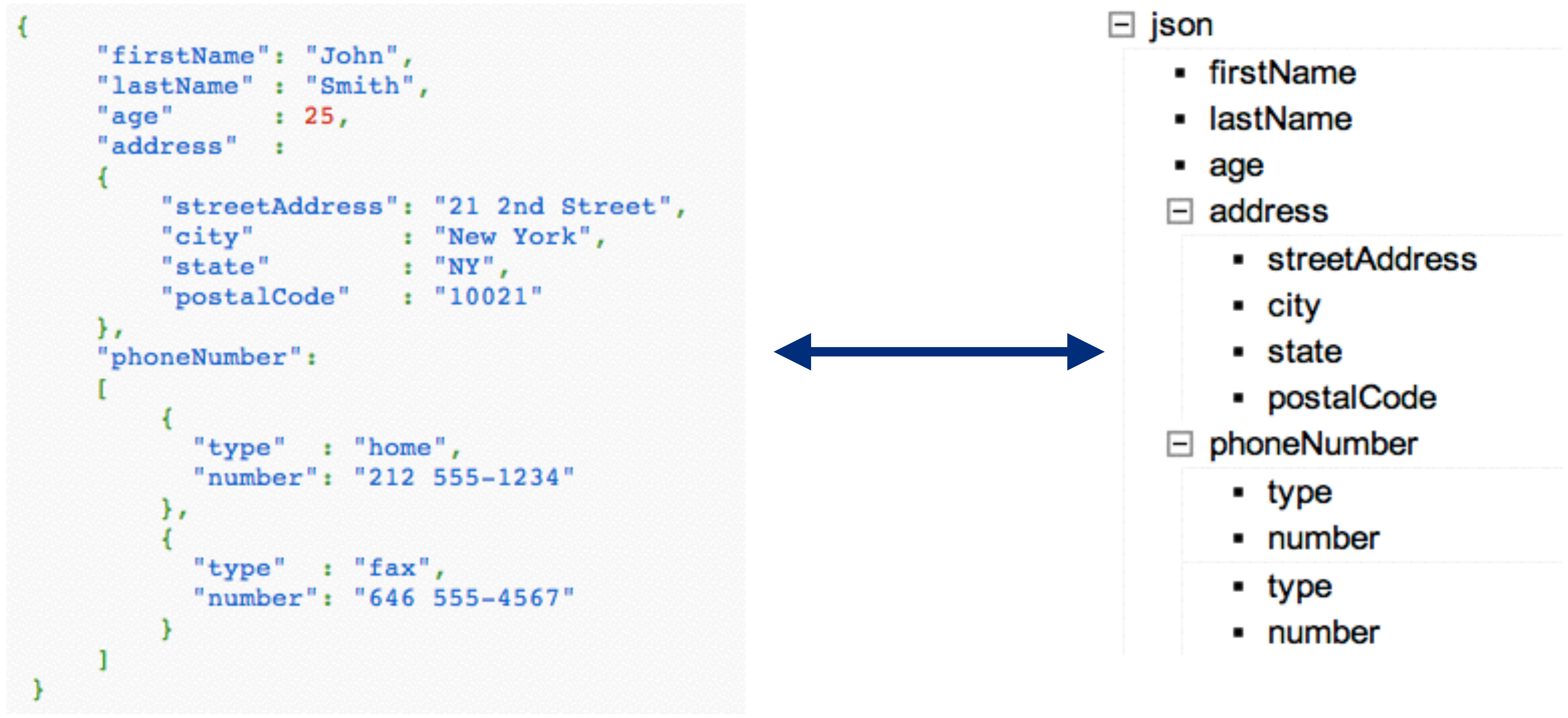
Reading XML Files

- The Android SDK includes several utilities for working with XML files, including SAX and XML Pull Parser, and limited DOM, Level 2 Core support.
- In particular useful packages are:

Method	Purpose
android.sax.*	Framework to write standard SAX handler.
android.util.Xml.*	XML utilities including the XMLPullParser
org.xml.sax.*	Core SAX functionality
javax.xml.*	SAX and limited DOM, Level 2 Core support
org.w3c.dom	Interface for DOM, Level 2 Core
org.xmlpull.*	XmlPullParser and XMLSerializer interfaces as well as a SAX2 Driver class

JSON

- JSON, or JavaScript Object Notation, is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for most languages.



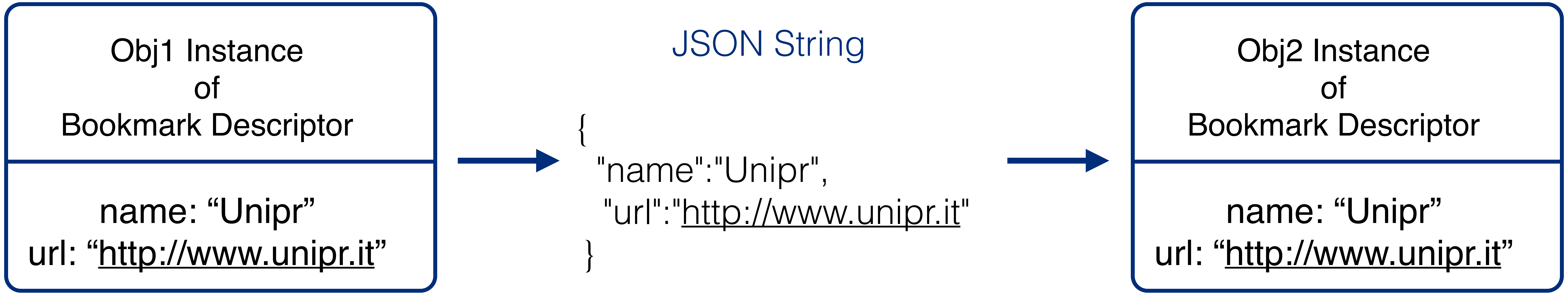
```
{
     "firstName": "John",
     "lastName" : "Smith",
     "age"      : 25,
     "address"  :
     {
         "streetAddress": "21 2nd Street",
         "city"         : "New York",
         "state"        : "NY",
         "postalCode"   : "10021"
     },
     "phoneNumber":
     [
         {
           "type"  : "home",
           "number": "212 555-1234"
         },
         {
           "type"  : "fax",
           "number": "646 555-4567"
         }
     ]
 }
```

```
⊟ json
  ▪ firstName
  ▪ lastName
  ▪ age
  ⊟ address
    ▪ streetAddress
    ▪ city
    ▪ state
    ▪ postalCode
  ⊟ phoneNumber
    ▪ type
    ▪ number
    ▪ type
    ▪ number
```

http://braincast.nl/samples/jsoneditor/

JSON

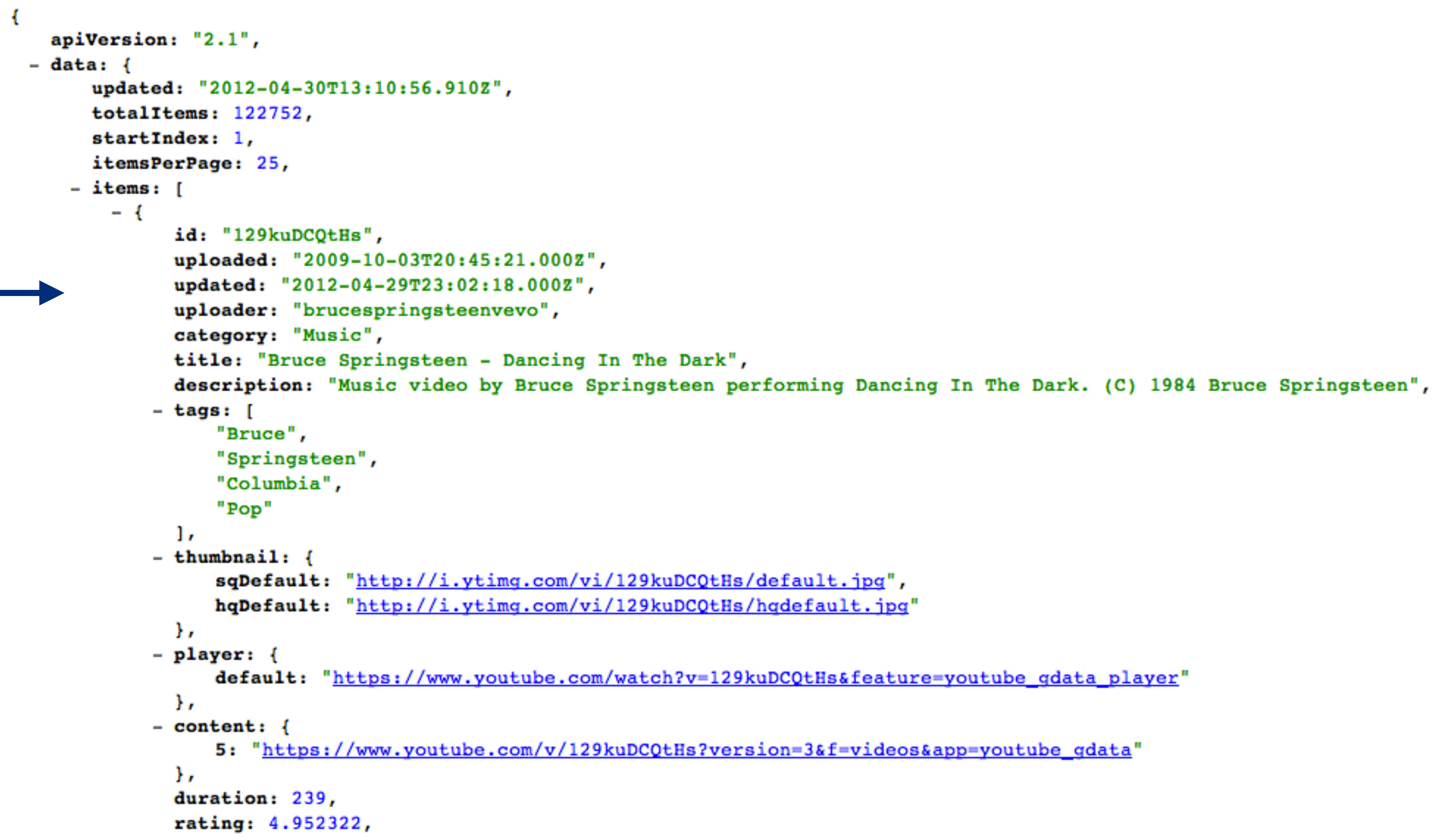
```java
public class BookmarkDescriptor {
   private String name = null;
   private String url = null;

   public BookmarkDescriptor()
   {}
   public BookmarkDescriptor(String name, String url)
    {
      this.name = name;
      this.url = url;
   }
}
```


Obj1 Instance of Bookmark Descriptor
name: “Unipr”
url: “http://www.unipr.it”
JSON String
Obj2 Instance of Bookmark Descriptor
name: “Unipr”
url: “http://www.unipr.it”

```
{
 "name":"Unipr",
  "url":"http://www.unipr.it"
 }
```

JSON

https://gdata.youtube.com/feeds/api/videos?q=Bruce%20Springsteen&v=2&alt=jsonc



```
{
    apiVersion: "2.1",
  - data: {
        updated: "2012-04-30T13:10:56.910Z",
        totalItems: 122752,
        startIndex: 1,
        itemsPerPage: 25,
      - items: [
          - {
                id: "129kuDCQtHs",
                uploaded: "2009-10-03T20:45:21.000Z",
                updated: "2012-04-29T23:02:18.000Z",
                uploader: "brucespringsteenvevo",
                category: "Music",
                title: "Bruce Springsteen - Dancing In The Dark",
                description: "Music video by Bruce Springsteen performing Dancing In The Dark. (C) 1984 Bruce Springsteen",
              - tags: [
                    "Bruce",
                    "Springsteen",
                    "Columbia",
                    "Pop"
                ],
              - thumbnail: {
                    sqDefault: "http://i.ytimg.com/vi/129kuDCQtHs/default.jpg",
                    hqDefault: "http://i.ytimg.com/vi/129kuDCQtHs/hqdefault.jpg"
                },
              - player: {
                    default: "https://www.youtube.com/watch?v=129kuDCQtHs&feature=youtube_gdata_player"
                },
              - content: {
                    5: "https://www.youtube.com/v/129kuDCQtHs?version=3&f=videos&app=youtube_gdata"
                },
                duration: 239,
                rating: 4.952322,
```

[...]

GSON Library

- 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.
- GSON:
  - Provide easy to use mechanisms like toString() and constructor (factory method) to convert Java to JSON and vice-versa
  - Allow pre-existing unmodifiable objects to be converted to and from JSON
  - Allow custom representations for objects
  - Support arbitrarily complex object
  - Generate compact and readability JSON output

GSON Library

- The primary class to use is Gson which you can just create by calling new Gson(). There is also a class GsonBuilder available that can be used to create a Gson instance with various settings like version control and so on.
- The Gson instance does not maintain any state while invoking Json operations. So, you are free to reuse the same object for multiple Json serialization and deserialization operations.

Primitives Examples

(Serialization)

```
Gson gson = new Gson();
gson.toJson(1);              ==> prints 1
gson.toJson("abcd");         ==> prints "abcd"
gson.toJson(new Long(10));   ==> prints 10
int[] values = { 1 };
gson.toJson(values);         ==> prints [1]
```

(Deserialization)

```
int one = gson.fromJson("1", int.class);
Integer one = gson.fromJson("1", Integer.class);
Long one = gson.fromJson("1", Long.class);
Boolean false = gson.fromJson("false", Boolean.class);
String str = gson.fromJson("\"abc\"", String.class);
String anotherStr = gson.fromJson("[\"abc\"]", String.class);
```

GSON Library

Object Examples

```
class BagOfPrimitives {
  private int value1 = 1;
  private String value2 = "abc";
  private transient int value3 = 3;
  BagOfPrimitives() {
    // no-args constructor
  }
}
```

(Serialization)

```
BagOfPrimitives obj = new BagOfPrimitives();
Gson gson = new Gson();
String json = gson.toJson(obj);
```

$\longrightarrow$ **{"value1":1,"value2":"abc"}**

(Deserialization)

```
BagOfPrimitives obj2 = gson.fromJson(json, BagOfPrimitives.class);
```

$\longrightarrow$ **obj2 is just like obj**

https://sites.google.com/site/gson/gson-user-guide

SQLite Database

- If your Android application requires a more solid data storage solution, the Android platform includes and provides the support for application-specific relational databases using SQLite.
- Any databases you create will be accessible by name to any class in the application, but not outside the application.
- SQLite database is a lightweight and file-based solution that stripped out features that are not absolutely necessary in order to achieve a small footprint.
- It has been designed to manage many kinds of system failures, such as low memory, disk errors, and power failures.
- The SQLite Project is not a Google Project, but an independent project with an international team of software developers.

SQLite Database

- SQLite in based on SQL language. SQL (Structured Query Language) is a programming language designed to manage data in relational database management systems (RDBMS).
- In this lecture we will use some basic commands to create and modify table and read and delete data in those tables.
- Detailed information and documentation at:
    - http://www.sqlite.org/
    - http://www.sqlite.org/lang.html (You can find a complete guide with more information about SQLite language)

Android Database Library

- The Android SDK includes a number of useful SQLite database management classes. Many of these classes are located in the android.database.sqlite package. Main classes are:
  - **SQLiteDatabase**: Java interface to SQLite. It support a SQL implementation rich enough for anything you will need in a mobile application.
  - **Cursor**: A container for the results of a database query to supports an MVC-style system. Using a Cursor it is possible to navigate the query results accessing each row when it is needed. Main methods are:
    - Cursor.getAs*(int columnNumber) (e.g. getAsString) to access available data.
    - Cursor.moveToNext or Cursor.moveToPrevious to change the current cursor index.
  - **SQLiteOpenHelper**: Provides a life cycle framework for creating and upgrading your application database.
  - **SQLiteQueryBuilder**: Provides a high-level abstraction for creating SQLite queries for use in Android applications. Using this class it is possible to simplify the task of writing queries.

Database Creation

- The recommended approach to create a new SQLite database on Android is to create a subclass of **SQLiteOpenHelper** and override the onCreate() and onUpgrade() methods.
- onCreate() is automatically called when the application starts for the first time creating the database.
- When new versions of the application are deployed, it may be necessary to update the database adding for example new tables, columns or changing the entire schema. When this is necessary, the task is associated with the onUpgrade() method, which is called whenever the DATABASE_VERSION in the call to the constructor is different from the one stored with the database.
- When you ship a new version of the database you must increment the version number in order to allow the real update.

Database Creation

```
public class LogDescriptorOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_NAME = "log.db";

    public static final String TABLE_NAME = "log";

    public static final String TIMESTAMP_COL = "timestamp";
    public static final String LATITUDE_COL = "latitude";
    public static final String LONGITUDE_COL = "longitude";
    public static final String TYPE_COL = "type";
    public static final String DATA_COL = "data";
    public static final String ID_COL = "id";

    private static final String DATABASE_TABLE_CREATE =
            "CREATE TABLE " + TABLE_NAME + " (" +
            ID_COL + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            TIMESTAMP_COL + " TIMESTAMP, " +
            LATITUDE_COL + " DOUBLE,"+
            LONGITUDE_COL + " DOUBLE,"+
            TYPE_COL + " VARCHAR(50),"+
            DATA_COL + " TEXT"+
            ");";
```

Query to create the table of our database.

```
public LogDescriptorOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        Log.d(LoggerAppActivity.TAG, "LogOpenHelper Constructor !");
    }

@Override
public void onCreate(SQLiteDatabase db) {
    Log.d(LoggerAppActivity.TAG,"LogOpenHelper onCreate !");
    db.execSQL(DATABASE_TABLE_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.d(LoggerAppActivity.TAG,"LogOpenHelper onUpgrade !");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    db.execSQL(DATABASE_TABLE_CREATE);
}

}
```

Use the query to initiate the database or upgrade it when it is necessary. Other upgrade behaviors can be implemented according to database type for example to save data that are already stored.

Database Creation

- The general elements associated with an SQLiteOpenHelper are:
  - DATABASE_NAME: This hold the filename of the database. This names the actual SQLite database file. Android platform will take care to create the database file for you in /data/data/<package_name>/databases/DATABASE_NAME.
  - DATABASE_VERSION: Defines the database version, which you choose arbitrarily and increment whenever you change the database schema. If the version of the database on the machine is less than DATABASE_VERSION, the system automatically runs onUpgrade() method to upgrade the database to the current level.
  - TABLE_NAME: The name of the table the we will create in our simple database.
  - Constructor: The constructor for the database. Uses the super function to call it's parent's constructor that will create the database object.
  - onCreate: When an Android application tries to read or write data to a database that does not exist, the framework executes the onCreate() method to initialize the database. This method will contain your code for the database initialization and tables creation.

SQLite Database Interaction

- You can execute SQLite queries using the SQLiteDatabase query() methods, which accept various query parameters, such as the table to query, the projection, selection, columns, grouping, and others.
- For complex queries, such as those that require column aliases, you should use SQLiteQueryBuilder, which provides several convenient methods for building queries.
- Every SQLite query will return a Cursor that points to all the rows found by the query. The Cursor is always the mechanism with which you can navigate results from a database query and read rows and columns.
- To add a new record in a table you can use the insert(...) method specifying the table to insert the row into and a map with column values for the row represented with an object called ContetValues.
- You can remove records from the database using the remove(...) method. It takes as arguments the table to delete the record from and two arguments describing the WHERE conditions.

Database Connection

- In order to interact with the database you should obtain an SQLiteDatabase instance through your OpenHelper subclass.
- The following code shows a potential application flow to retrieve the object, work with the DB and then close the connection.

```
{

  LogDescriptorOpenHelper dbHelper = new LogDescriptorOpenHelper(context);
  SQLiteDatabase database = dbHelper.getWritableDatabase();

 ... DB Operations ...

  dbHelper.close();
}
```

DB Query

```
public List<LogDescriptor> getAllLogsDescriptor() {

   String[] allColumns = { LogDescriptorOpenHelper.ID_COL,LogDescriptorOpenHelper.TIMESTAMP_COL,
   LogDescriptorOpenHelper.LATITUDE_COL, LogDescriptorOpenHelper.LONGITUDE_COL,
   LogDescriptorOpenHelper.TYPE_COL, LogDescriptorOpenHelper.DATA_COL };

      List<LogDescriptor> logList = new ArrayList<LogDescriptor>();

      Cursor cursor = database.query(LogDescriptorOpenHelper.TABLE_NAME,allColumns, null,null, null, null,null);
      cursor.moveToFirst();

      while (!cursor.isAfterLast()) {
         LogDescriptor logDescr = cursorToLogDescriptor(cursor);
         comments.add(logDescr);
         cursor.moveToNext();
      }

      // Make sure to close the cursor
      cursor.close();
      return comments;
   }
```

Query columns.

Additional Args:
selectionArgs
groupBy
having
orderBy
limit

Use the Cursor obtained object to parse results.

DB Query

```
private LogDescriptor cursorToLogDescriptor(Cursor cursor) {
     LogDescriptor logDescriptor = new LogDescriptor();
     logDescriptor.setId(cursor.getInt(0));
     logDescriptor.setTimestamp(cursor.getLong(1));
     logDescriptor.setLatitude(cursor.getLong(2));
     logDescriptor.setLongitude(cursor.getLong(3));
     logDescriptor.setType(cursor.getString(4));
     logDescriptor.setData(cursor.getString(5));
     return logDescriptor;
  }
```

Read values from the Cursors
(with **cursor.get*(<column_index>)**) to build a LogDescriptor object.

Insert & Query

```
ContentValues values = new ContentValues();
values.put(LogDescriptorOpenHelper.TIMESTAMP_COL, log.getTimestamp());
values.put(LogDescriptorOpenHelper.LATITUDE_COL, log.getLatitude());
values.put(LogDescriptorOpenHelper.LONGITUDE_COL, log.getLongitude());
values.put(LogDescriptorOpenHelper.TYPE_COL, log.getType());
values.put(LogDescriptorOpenHelper.DATA_COL, log.getData());
```

Create a ContentValues object to set values for each table column. (If a value is not set it will be automatically set to NULL).

```
long insertId = database.insert(LogDescriptorOpenHelper.TABLE_NAME, null,values);
```

Insert configured values into the desired table. The result values is the raw ID of the new record (-1 if there is an error).

```
Cursor cursor = database.query(LogDescriptorOpenHelper.TABLE_NAME, allColumns,
                LogDescriptorOpenHelper.ID_COL + " = " + insertId, null, null, null, null);
```

Query the DB to obtain the object of the last added record. Use the query(...) method specifying the WHERE condition.

```
cursor.moveToFirst();
LogDescriptor obj = cursorToLogDescriptor(cursor);
```

Read values from the Cursor and build the object.

Delete

```
public void deleteAllLogs() {
   database.delete(LogDescriptorOpenHelper.TABLE_NAME,null, null);
}
```

Delete all records in the specified table without particular WHERE conditions.

```
public void deleteLog(LogDescriptor logDescr) {
      long id = logDescr.getId();
      database.delete(LogDescriptorOpenHelper.TABLE_NAME,LogDescriptorOpenHelper.ID_COL + " = " + id, null);
   }
```

Delete all records in the specified table with a particular WHERE condition. In this case it will delete the record with a specific ID.

Raw Query

```
int logCount = 0;

String query = "SELECT COUNT(*) FROM "+LogDescriptorOpenHelper.TABLE_NAME;

Cursor cursor = database.rawQuery(query,null);

cursor.moveToFirst();

...

cursor.close();
```

You can run raw queries using SQL language and the dedicated method rawQuery(...). In this case the query is built by the developer as a String and passed to the function as main parameter.

I this example the query return the number of records in the table.

Android Development

Lecture 6
Data Persistence

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.