Lesson 4 - Characters App (Parts 1 & 2)

Objectives of Part 1 & 2

- Describe SQL commands to create a SQLite database table, insert rows into a database table, query a database table and delete a row
- Write a helper class that extends SQLiteOpenHelper to carry out the tasks of creating, inserting, querying and deleting for a SQLite database
- Make use of the helper class to display a random image
- Use explicit intents and implicit intents to write a UI for the user to provide data to insert a row into the database

The Android/Java that you need to know

Recall Lesson 2 & 3

We might get bogged down with specific details in the code, but we should also gain an overview of the Android framework.
For each feature described, state the part of the android framework that you can implement.

Feature	**Framework component**
Bring the user from one Activity to another Activity	
To validate that your app behaves as it should when a user interacts with it	
Bring the user from the app to another app with a specific function	
The series of callbacks as an Activity is created and/or destroyed	
To run long tasks in another thread	
Store data to make it available at another point in time	
To validate that code components (e.g. methods) in your app behaves as it should	

Static Nested Classes

Recall that a class definition can contain **nested classes**.

```
public class OuterClass {
   // code not shown

   class InnerClass{
       //code not shown
   }

}
```

This is typically done when you have classes that logically depend on the outer class and are used together with the outer class.

By declaring a nested class as static, it is known as a **static nested class.**

- It can only access static variables and methods in the outer class.
- It can be instantiated without an instance of the outer class.

A static nested class behaves like a top-level class and is a way to organize classes that are used only by some other classes.
One reason for having a static nested class is to have a model class to store data.

Recall the Singleton design pattern

Recall that the singleton design pattern allows only one instance of a class to exist.

This is done by

- Making the constructor private
- The sole instance is stored in a private static variable
- Using a static factory method to return an instance

```
public class Singleton{

    private static Singleton singleton;

    private Singleton(){
         //any tasks you need to do here
    }

    public Singleton getInstance(){

        if(singleton == null){
            singleton = new Singleton();
        }

        return singleton;
     }

     //other methods in your class

}
```

Getting Used to SQL Commands

A **relational database** (“**SQL database**”) has its data stored in tables. The closest thing you might be familiar with is having data in a spreadsheet.

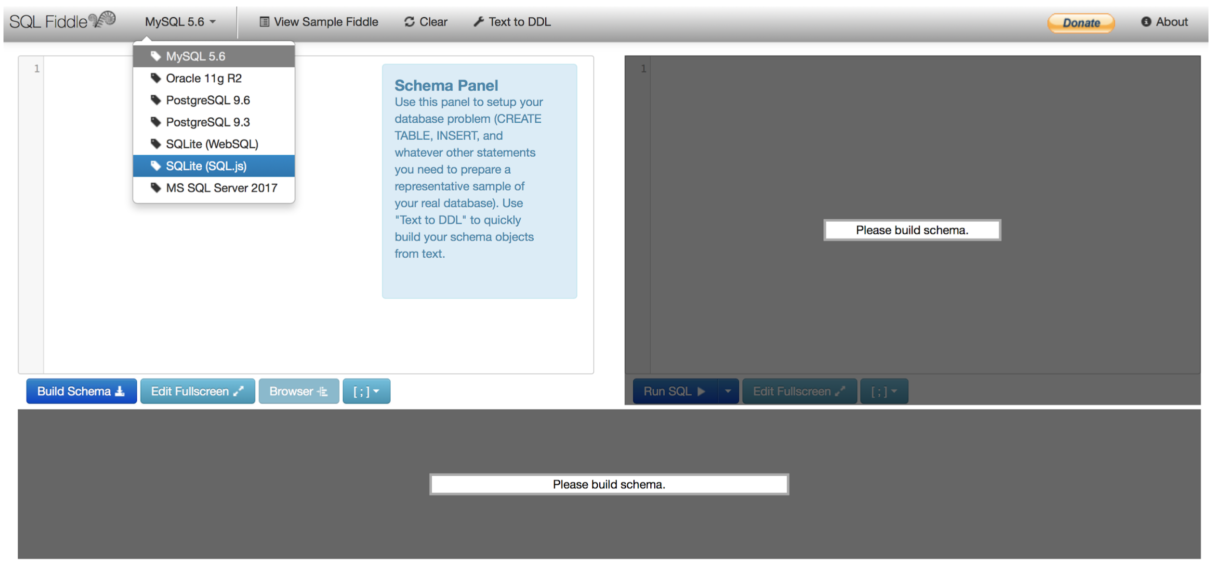
A **non-relational database** (“**noSQL database**”) has its data stored in other ways. One common way is using key-value pairs, which you would have encountered in Google firebase.

In this section, you are going to be introduced to the SQL language to query a **relational database**. We will be writing an android app to query a local SQLite database, which is available for all apps to use.

A relational database will have more than one table. However, to illustrate the essential concepts to you, we will just create one table and query it.

Step 1. Access http://sqlfiddle.com .
If you are a Mac user, you may check if your terminal has sqlite by typing in **`sqlite3`**.

Step 2. Click on **MySQL5.6** and from the dropdown menu, select **SQLite (SQL.js)**. Ensure that the box beside “Browser” shows a semi-colon.



Schema Panel
Use this panel to setup your database problem (CREATE TABLE, INSERT, and whatever other statements you need to prepare a representative sample of your real database). Use "Text to DDL" to quickly build your schema objects from text.

Please build schema.

Please build schema.

Step 3. Put in *all* the following commands in the left hand side box and click **Build Schema**. The response from the website might be slow, wait for a while and try again if you are unsuccessful.
The first line creates a table with three columns (what are the column names?) and the subsequent three lines insert rows into the table.

```
CREATE TABLE SpendingRecord (
  _ID INTEGER PRIMARY KEY AUTOINCREMENT,
Amount INTEGER NOT NULL,
Remarks TEXT NOT NULL );

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (10000, 'Man');

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (20000, 'Norman');

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (30000, 'Eric');
```

Let's unpack the command to create a table.

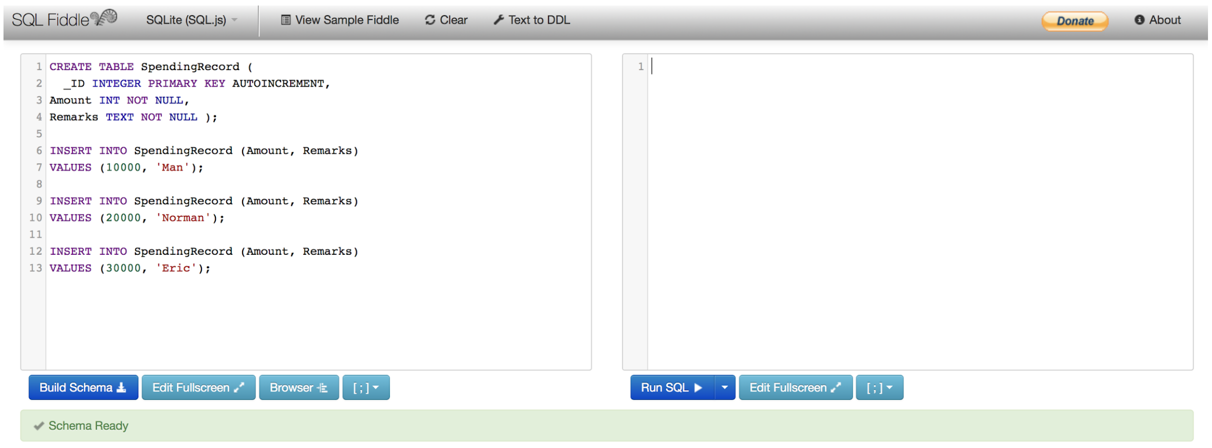
The table name is called **`SpendingRecord`**. There are three columns in this table, called **`_ID`**, **`Amount`** and **`Remarks`**.

The column called **`_ID`** is specified as a **`PRIMARY KEY`** and is of **`INTEGER`** (integer) data type. A **primary key** is a unique entry that identifies each row in the table. Your student ID is a primary key. The primary key need not be an integer.
`AUTOINCREMENT` specifies that as each row is added, the primary key increases by one.

Subsequently, the column called **`Amount`** is specified to be of Integer type and **`NOT NULL`** specifies that the rows in the column are not allowed to be empty.
Finally, the column called **`Remarks`** is specified to be of Text data type i.e. it accepts strings.

It is possible for the table to store images. The datatype is called **`BLOB`**.

Step 4. If you are successful, you should see the following screen. You are ready to key in commands to query your database on the right-hand panel.



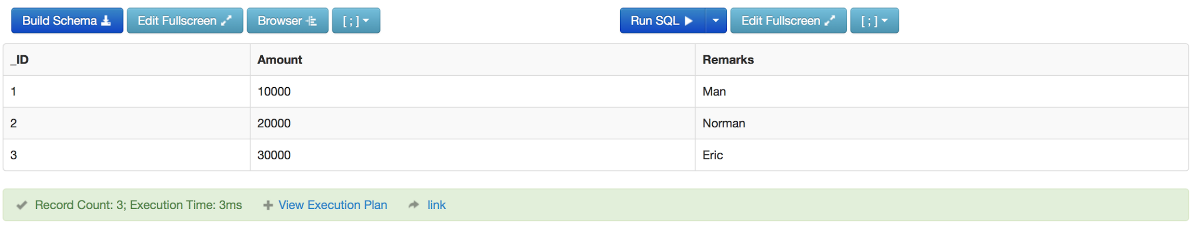
```
CREATE TABLE SpendingRecord (
  _ID INTEGER PRIMARY KEY AUTOINCREMENT,
Amount INT NOT NULL,
Remarks TEXT NOT NULL );

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (10000, 'Man');

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (20000, 'Norman');

INSERT INTO SpendingRecord (Amount, Remarks)
VALUES (30000, 'Eric');
```

Step 5. Run some SQL queries.

Here is the command to query the entire table. Put it in the right-hand box and click **Run SQL**.

```
SELECT * FROM SpendingRecord;
```

You should see this appear below:



_ID	Amount	Remarks
1	10000	Man
2	20000	Norman
3	30000	Eric

Record Count: 3; Execution Time: 3ms   View Execution Plan   link

If you wish to see only a particular column, specify the column name after **`SELECT`**. :

```
SELECT Remarks FROM SpendingRecord;
```

You may use a “WHERE clause” to query subsets of your table. For example:

Access the row where Remarks contains the text Eric:

```
SELECT * FROM SpendingRecord WHERE Remarks = 'Eric';
```

Access the rows where amount > 15000

```
SELECT * FROM SpendingRecord WHERE Amount > 15000;
```

Access a random row:

```
SELECT * FROM SpendingRecord ORDER BY RANDOM() LIMIT 1;
```

Further reading

Android Developer Fundamentals v2, Concept Reference, Section 10.0
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-4-saving-user-data/lesson-10-storing-data-with-room/10-0-c-sqlite-primer/10-0-c-sqlite-primer.html

We learnt about Data Persistence using **`SharedPreferences`**.
Data is stored using **key-value pairs**.
Hence, if there is a lot of data, using **`SharedPreferences`** will be tedious.
An alternative is a local **SQLite database** on the phone within an android app.

To create and manage a local SQLite database in your android app,
write a class that subclasses **`SQLiteOpenHelper`**.

This class will perform the following tasks

- Create the database - override **`onCreate()`**
- Upgrade the database - override **`onUpgrade()`**

This class can also include methods for the database CRUD queries

- **C**reating records
- **R**eading or Querying records
- **U**pdating records
- **D**eleting records

How to use SQLiteOpenHelper

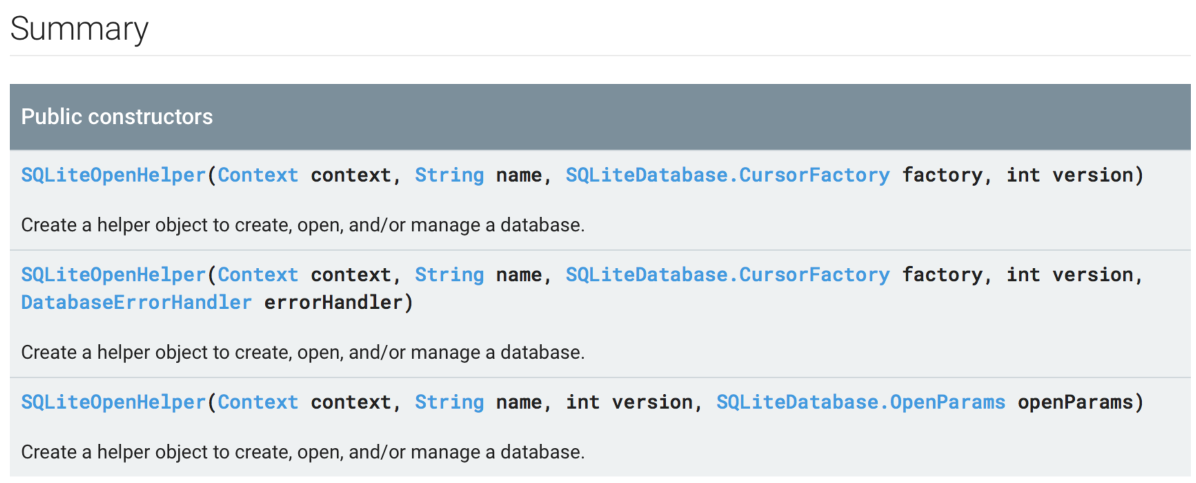
Constructor

Your constructor should take in a Context object.

Then, call the superclass constructor with

- The same context object
- The table name
- null for the CursorFactory
- The database version (explained below).

An extract from the documentation is shown below.



Summary

Public constructors
`SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)` Create a helper object to create, open, and/or manage a database.
`SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)` Create a helper object to create, open, and/or manage a database.
`SQLiteOpenHelper(Context context, String name, int version, SQLiteDatabase.OpenParams openParams)` Create a helper object to create, open, and/or manage a database.

The **singleton design pattern** should also be used with this class,

as only one instance of the helper should be querying the database at any point in time.

Database Version

Included in the superclass constructor is an integer variable **`version`**

that is usually initialized to 1. This is the database version.

Usually, this is declared as a static variable in the class and passed to the constructor.

You manually increment this number when you make changes to the database e.g.

- After adding/removing columns
- After fixing a mistake in your SQL commands

Incrementing this number triggers a call to **`onUpgrade()`**.

`onCreate()`

In onCreate, an instance of a **`SQLiteDatabase`** object is passed to it.
Execute its **`execSQL`** method by passing to it the SQL command for creating a table.
You may also write code

- to populate this database with initial entries
- to display logcat messages to see when it is executed.

`SQL_CREATE_TABLE` is a static string variable that stores an SQL command to create the table in the schema that you desire.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
   sqLiteDatabase.execSQL(SQL_CREATE_TABLE);
   fillTable(sqLiteDatabase);
}
```

`onUpgrade()`

This method is triggered when the database version is incremented.

- The SQL command to delete the table is executed.
- onCreate is then invoked

`SQL_DROP_TABLE` is a static string variable that stores an SQL command to delete the table.

```
@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int
i1) {
   sqLiteDatabase.execSQL(SQL_DROP_TABLE);
   onCreate(sqLiteDatabase);
}
```

SQLiteDatabase Operations - Querying

You may write methods to query the SQLite database in the helper.

How to start

Begin by calling **`getReadableDatabase()`**, which returns an **`SQLiteDatabase`** object.

Usually the result is stored in an instance variable.

Then execute an SQL query using the **`rawQuery`** method.

The raw query method returns a **`Cursor`** object with the result

```
if (readableDb == null){
   readableDb = getReadableDatabase();
}
String SQLCommand = ; //command is written here
Cursor cursor = readableDb.rawQuery(SQLCommand, null);
```

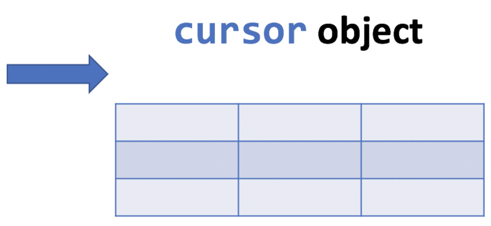
Using the cursor object

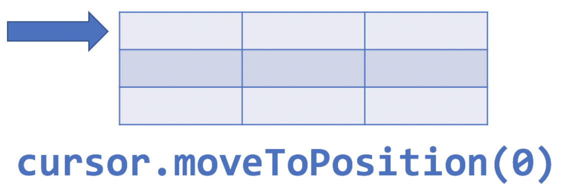
Depending on the SQL query, the number of columns returned could vary. There could also be more than one row.

- The cursor object will point initially to outside of the table,
  you need to move the pointer to the row you want.
  Use the **`moveToPosition()`** method. The first row is position = 0.
- Obtain the column index in the cursor object using the column name.
  In the example below, a static string variable **`COL_NAME`** is used for this.
- Use the column index to retrieve the data.

Repeat the code for each column that you wish to retrieve.
Regardless of the number of columns, **`moveToPosition()`** needs to be called only once per row.

```
cursor.moveToPosition(position);
int nameIndex = cursor.getColumnIndex(COL_NAME);
String name = cursor.getString(nameIndex);
```





When the `cursor` object is first returned by the database query, the arrow points to outside the table.

Invoking `moveToPosition(0)` points the arrow to the 0th row.

SQLiteDatabase Operations - Creating New Rows

To create a new row, you need not run any SQL query.

Android has the following solution:

- Begin by calling **`getWritableDatabase()`**, which returns an **`SQLiteDatabase`** object. Usually the result is stored in an instance variable.
- Create a new **`ContentValues`** object.
- Use the **`put`** method on this object to put the data, with the column name as a key.
- Finally, call the **`insert`** method on the SQLiteDatabase object and provide the table name and the ContentValues object as inputs.

The code sample below shows you how to insert data into a table with one column.

```
 if (writeableDb == null){
   writeableDb = getWritableDatabase();
}
ContentValues cv = new ContentValues();
cv.put(COL_NAME, nameData);
writeableDb.insert(TABLE_NAME,null,cv);
```

SQLiteDatabase Operations - Deleting a Row

To delete a row, you only need the **where-clause** part of the SQL query.
Android has the following solution:

- Begin by calling **`getWritableDatabase()`**, which returns an **`SQLiteDatabase`** object. Usually the result is stored in an instance variable.
- Write your where-clause as a string. Notice what is placed after the equal sign.
- Arguments that replace the question mark are stored in a string array.
- Finally, call the **`delete`** method on the SQLiteDatabase object and provide the table name, where-clause and where-args. This method returns the number of rows deleted.

```
if (writeableDb == null){
   writeableDb = getWritableDatabase();
}

String WHERE_CLAUSE = COL_NAME + " = ?";
String[] WHERE_ARGS = {name};
int rowsDeleted =
writeableDb.delete(TABLE_NAME,WHERE_CLAUSE,WHERE_ARGS);
```

https://guides.codepath.com/android/local-databases-with-sqliteopenhelper

Utility Classes

You will realize that there are several **string constants** that are used throughout the code.
These include

- table name and column names
- SQL commands

Placing such information in **static variables**
within **static inner classes declared final** can help to organize it.

As the function of such classes is just to store data,
instantiation ought to be prevented by making the constructor private.

Lastly, **having static inner classes groups related classes together.**

An *excerpt* (due to space constraints) of the such a class in this lesson is shown below.

```
public class CharaContract {

   public static final class CharaEntry implements BaseColumns {
       public static final String TABLE_NAME = "Chara";
       public static final String COL_NAME = "name";
       public static final String COL_DESCRIPTION = "description";
       public static final String COL_FILE = "file";
   }

   public static final class CharaSql {

       public static String SQL_DROP_TABLE = "DROP TABLE IF EXISTS
" + CharaEntry.TABLE_NAME;
       public static String SQL_QUERY_ALL_ROWS = "SELECT * FROM "
+ CharaEntry.TABLE_NAME;
   }
}
```

Examples of how variables are accessed as follows. Notice how the use of inner classes can add to making the names meaningful.

```
CharaContract.CharaEntry.TABLE_NAME
CharaContract.CharaSql.SQL_DROP_TABLE
```

Static inner classes can also be used to model data

An inner class declared in the database helper class mimics the structure of each row of the table, and helps in passing data around.
Bearing in mind that such static classes cannot access non-static variables of the enclosing class.

```
public class CharaDbHelper extends SQLiteOpenHelper {

//code not shown --

   static class CharaData{

       private String name;
       private String description;
       private String file;
       private Bitmap bitmap;

       //constructors and getters not shown

   }

}
```

Intents - startActivityForResult()

Recall that

- **Explict intents** bring you from one activity to another. Data can be passed during this process.
- **Implicit intents** bring you from one activity to a component in your app. You specify the kind of component you want, and the android runtime fetches what is available.

In both cases, you launched the “destination” by invoking startActivity().

By invoking **`startActivityForResult()`**,
you expect the destination component to return a result.

Explicit Intent - startActivityForResult()

Step 1. Declare your request code, a final static integer variable that contains a unique integer that identifies your particular intent.
This is necessary as your activity could have more than one call to **`startActivityForResult()`**

```
public static int REQUEST_CODE_FAB = 1000;
```

Step 2. Declare an explicit intent in the usual way.
Then invoke **`startActivityForResult()`** with two arguments

- The intent
- The request code

```
Intent intent = new Intent(MainActivity.this, DataEntryActivity.class);
startActivityForResult(intent, REQUEST_CODE_FAB);
```

Step 3. In the destination activity, the user should interact with it.

Step 4. A user action (e.g. clicking a button) brings the user back to the origin activity.
The following code initiates this process.

```
Intent returnIntent = new Intent();
returnIntent.putExtra(KEY, value); //optional
setResult(Activity.RESULT_OK, returnIntent);
finish();
```

The first argument of **`setResult()`** is either

- **`Activity.RESULT_OK`** if the user has successfully completed the tasks
- **`Activity.RESULT_CANCELED`** if the user has somehow backed out

Hence, this code may be written twice, one for each scenario described above.
If you have data to transfer, you are reminded that you can use the **`putExtra()`** method above. (Recall Lesson 2).

Step 5. Back in the origin activity, override the callback `onActivityResult()` to listen out for the result and carry out the next task. Note the sequence of if-statements.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {

   if (requestCode == REQUEST_CODE_FAB) {
       if(resultCode == Activity.RESULT_OK){

           //if you use putExtra in Step 4, then you need this step
           double value = data.getDoubleExtra(DataEntryActivity.KEY,
defaultValue);
           Toast.makeText(this,"Message",Toast.LENGTH_LONG).show();
       }
       if (resultCode == Activity.RESULT_CANCELED) {
           //Write your code if there's no result
       }
   }
}
```

If you have data transferred from step 4, you may retrieve this data on the intent object passed to this callback using the **`getDoubleExtra()`** method or other suitable methods(Recall Lesson 2).

Further reading

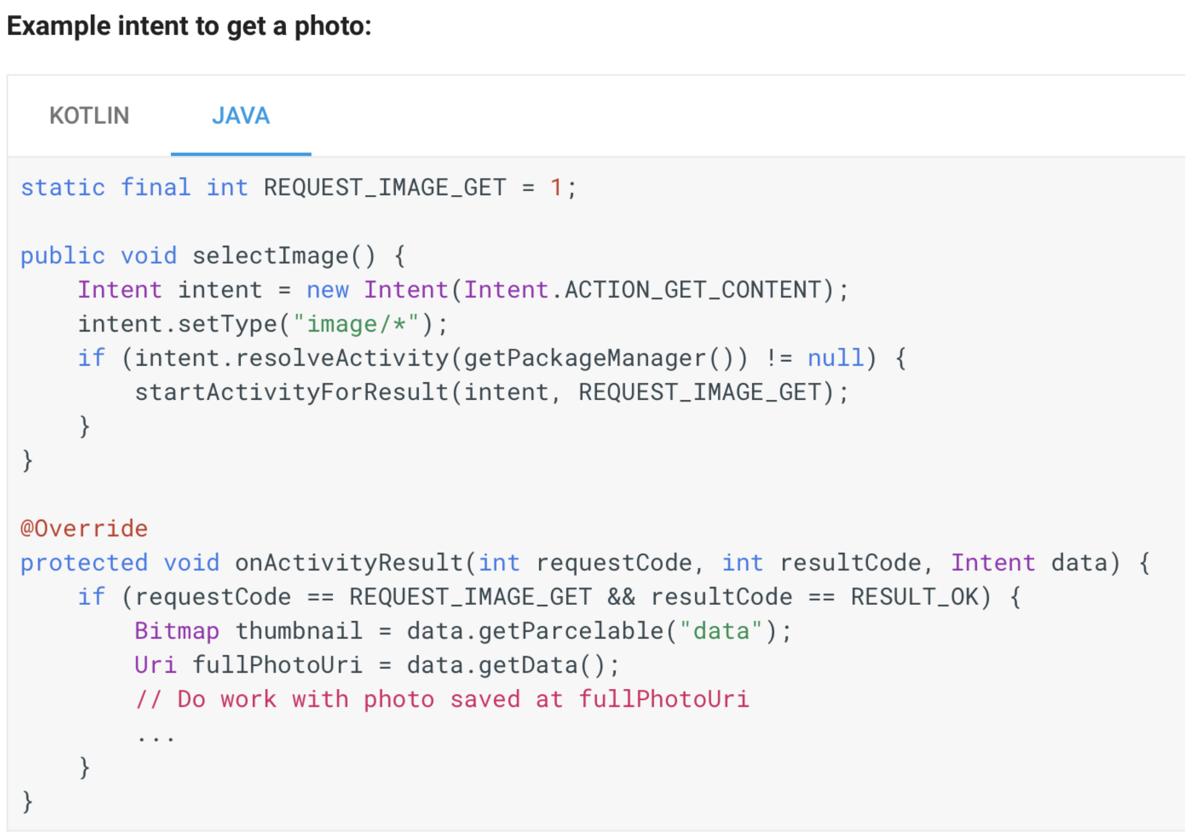
- https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-2-activities-and-intents/2-1-c-activities-and-intents/2-1-c-activities-and-intents.html#gettingdatabackfromactivity
- https://developer.android.com/training/basics/intents/result

Implicit Intents - opening the image gallery

Attention if you are using the android emulator. To have some images for you to select, one easy way is to use the emulator to take some pictures first. I leave you to discover the best solution for yourself.

The implicit intents to write can be found at the following **Common Intents** reference
https://developer.android.com/guide/components/intents-common#java
To get the example code to retrieve an image, go to the **File Storage → Retrieve a Specific Type of File** section.

In case the website is not accessible, here is the sample code,



Example intent to get a photo:

KOTLIN JAVA

```java
static final int REQUEST_IMAGE_GET = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_GET);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_GET && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        Uri fullPhotoUri = data.getData();
        // Do work with photo saved at fullPhotoUri
        ...
    }
}
```

In order to convert a URI object to a bitmap object, here's the code:
A static method is provided in the Utils class for converting an InputStream object to a Bitmap object.

```
InputStream inputStream =
this.getContentResolver().openInputStream(data.getData());
bitmapSelected = Utils.convertStreamToBitmap(inputStream);
```

Building Your App (Part 1)

Objective For Part 1

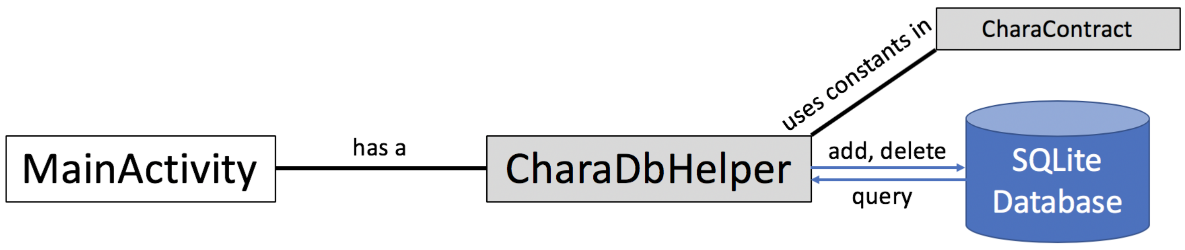
We will build CharaDbHelper with methods to

- Create the database
- Query the database
- Insert one row
- Delete one row

In this part, we will **not** be working with the UI.

All information will be printed to the Logcat.

The architecture of your app will be:



TODO 7.1 & 7.2 Examine the CharaContract Class and prevent instantiation of this class

The `CharaEntry` inner class stores the column names of the table.

The `CharaSql` inner class stores the SQL commands needed.

You may write logcat statements in `MainActivity` to display these strings.

TODO 7.3 Complete CharaData inner class with getters

The `CharaData` inner class is used in two ways:

- To store the name, description and filename as read from the pictures.json file in the raw folder
- To store the name, description and bitmap file

TODO 7.4 Make CharaDbHelper a singleton class

The constructor for `CharaDbHelper` is given to you. Implement the singleton design pattern with it.

TODO 7.5 & 7.6 Override and Implement onCreate() and onUpgrade()

TODO 7.7 Complete queryNumRows() to return the number of rows in the database

TODO 7.8 - 7.11 Complete the methods to conduct database operations

- getDataFromCursor()
- queryOneRowRandom()
- queryOneRow()
- insertOneRow()
- deleteOneRow()

TODO 7.12 & 7.13 In MainActivity, instantiate CharaDbHelper and test the methods written in TODO 7.7 - 7.11

Use the methods in TestCharaDbHelper to verify that your methods above do what they should.

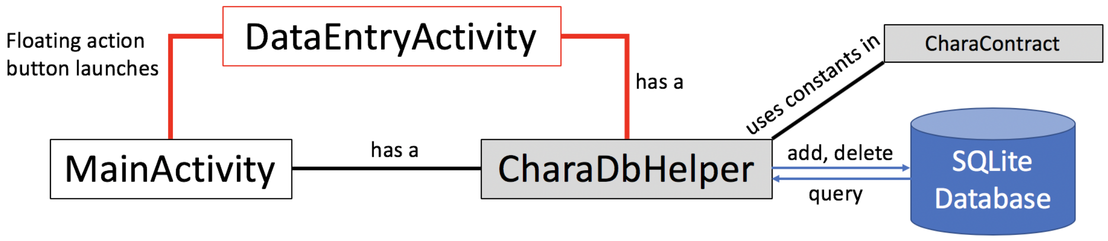
Building Your App (Part 2)

Objective For Part 2

Assuming that you have a working CharaDbHelper, then you are ready to make use of it in the user interface.

- Display a random image on the UI
- Obtain an image from your phone's image gallery and add it to your database

The architecture of your app will be:



TODO 8.1 & 8.2 When the button is clicked, query the database for a random image and display it on the screen

TODO 8.3 When the floating action button is clicked, bring the user to DataEntryActivity using startActivityForResult()

In DataEntryActivity ….

TODO 8.4 & 8.5 Get references to the widgets on the layout and CharaDbHelper

TODO 8.6 & 8.7 When the Select Image button is clicked, set up an implicit intent to the image gallery and complete onActivityResult() to receive the data

TODO 8.8 When the OK button is clicked, the data is added to the database and the user is brought back to MainActivity

In MainActivity …

TODO 8.9 Complete onActivityResult() to display a toast