



Android Boot Camp

Day 1 - Android and Databases

Ronald L. Ramos

December 2015

Ronald L. Ramos, Technology Group



Ronald L. Ramos

- rlramos@smart.com.ph
- 09298874076/ 5113175
- Twitter: @taleweaver
- Facebook: rlramos@gmail.com
- Blog: <http://brickstories.blogspot.com>
- Instagram: taleweaver



- Get the presentation here:

[http://mobiledev.ronaldramos.info/
androidbootcamp2016/](http://mobiledev.ronaldramos.info/androidbootcamp2016/)



Introduce yourself

- Name/Nickname
- Android Dev experience
- Expectations for the training
- Smartphone user? What's your smartphone?
- Favorite tv show? Why?



Topics

- History of Android
- Android Basics
- ListViews
- Creating DBs local to the device



What is Android?

Android is an open mobile phone platform that was developed by Google and later by Open Handset Alliance. Google defines Android as a "software stack" for mobile phones.

A software stack is made up of operating system(the platform on which everything runs), the middleware (the programming that allows applications to talk to a network and to one another) and the applications (the actual programs that phone will run)



Brief History

July 2005 - Google Inc. bought from Danger Inc

Open Handset Alliance was formed headed by Google which is composed of companies like Intel, T-Mobile, Spring Nextel and more.

In 2008, Android became available as an open source and the ASOP(Android Open Source Project) is responsible for maintainance and development of android.



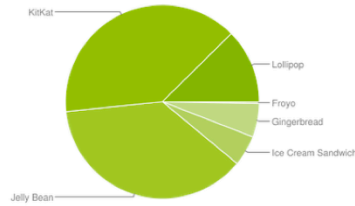
February 2009, the first android version was released

Android Versions

Version Number	Name
1.1	
1.5	Cupcake
1.6	Donut
2.0/2.1	Éclair
2.2.x	Froyo
2.3.x	Gingerbread
3.X	Honeycomb
4.0-4.0.4	Ice Cream Sandwich
4.1-4.3.1	Jelly Bean
4.4	KitKat
5	Lollipop
6	Marsh Mallow

Android Version Market Share

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%



Data collected during a 7-day period ending on June 1, 2015.
Any versions with less than 0.1% distribution are not shown.

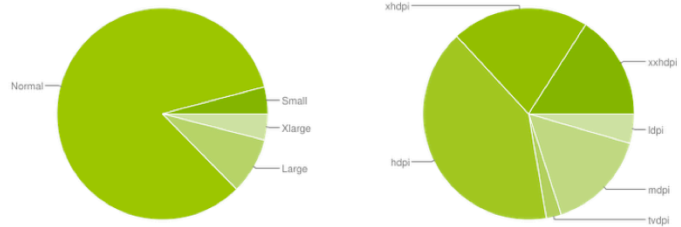
Note: When developing an application, consider the market share of the android version. The higher the market share, the higher number your target market is.



<https://developer.android.com/about/dashboards/index.html>

Screen Densities Distribution

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	4.1%						4.1%
Normal		7.6%	0.1%	39.9%	19.8%	15.9%	83.3%
Large	0.4%	4.8%	2.2%	0.6%	0.6%		8.6%
Xlarge		3.1%		0.3%	0.6%		4.0%
Total	4.5%	15.5%	2.3%	40.8%	21.0%	15.9%	



Data collected during a 7-day period ending on June 1, 2015.
 Any screen configurations with less than 0.1% distribution are not shown.



Why develop for Android?

- Android is an open-source platform based on the Linux kernel, and is installed on thousands of devices from a wide range of manufacturers.
- Android exposes your application to all sorts of hardware that you'll find in modern mobile devices — digital compasses, video cameras, GPS, orientation sensors, and more.
- Android's free development tools make it possible for you to start writing software at little or no cost.
- When you're ready to show off your application to the world, you can publish it to Google's Android Market. Publishing to Android Market incurs a one-off registration fee (US \$25 at the time of writing) and, unlike Apple's App Store which famously reviews each submission, makes your application available for customers to download and buy after a quick review process — unless the application is blatantly illegal.



More Android Advantages


- The Android SDK is available for Windows, Mac and Linux, so you don't need to pay for new hardware to start writing applications.
- An SDK built on Java. If you're familiar with the Java programming language, you're already halfway there.
- By distributing your application on Android Market, it's available to hundreds of thousands of users instantly. You're not just limited to one store, because there are alternatives, too. For instance, you can release your application on your own blog. Amazon have recently launched their own Android app store also.
- As well as the technical SDK documentation, new resources are being published for Android developers as the platform gains popularity among both users and developers.



SETTING UP THE ANDROID DEVELOPMENT ENVIRONMENT



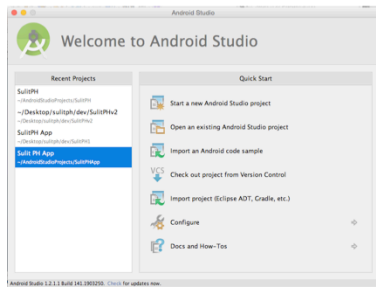
What you'll need

- 
- Computer running Windows, Linux, or Mac OS X
 - Android Studio (IDE with SDK)

The Bundle is found on developer.android.com



Android Studio



- <http://developer.android.com/tools/studio/index.html>
- Android Studio is the official IDE for Android application development, based on IntelliJ IDEA



<http://developer.android.com/tools/studio/index.html>

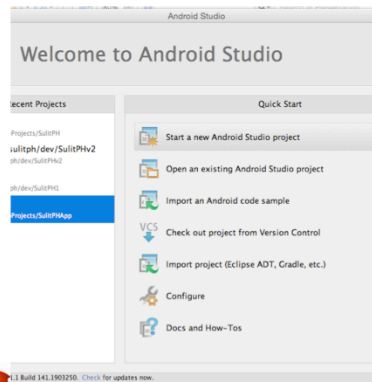
Creating your first project

HELLO WORLD



Creating a New Android Project

- Open Android Studio
- Select “Start a new Android Studio Project”



Configure project

Create New Project

New Project
Android Studio

Configure your new project

Application name:

Company Domain:

Package name: [Edit](#)

Project location:



Select Target Devices

Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms require separate SDKs

Phone and Tablet
Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

TV
Minimum SDK: API 21: Android 5.0 (Lollipop)

Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)

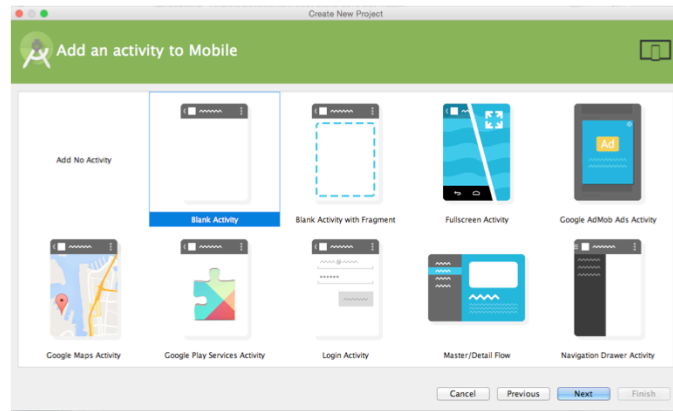
Class
Minimum SDK: Class Development Kit Preview (Google Inc.) (API 19)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately 90.4% of the devices that are active on the Google Play Store. [Help me choose.](#)

Cancel Previous **Next** Finish

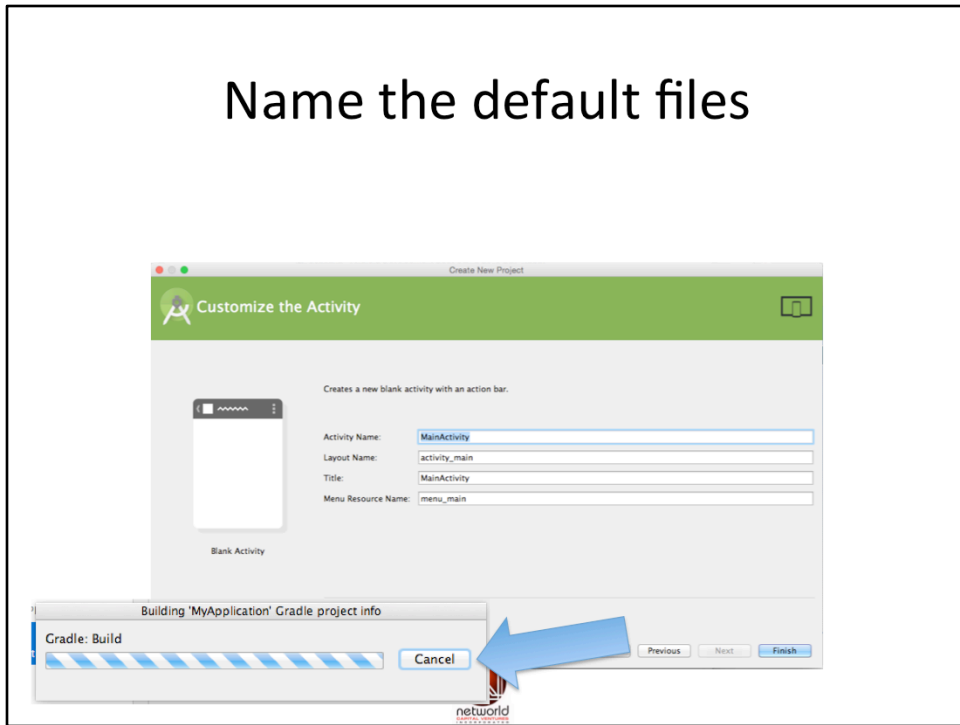


Select Activity Template

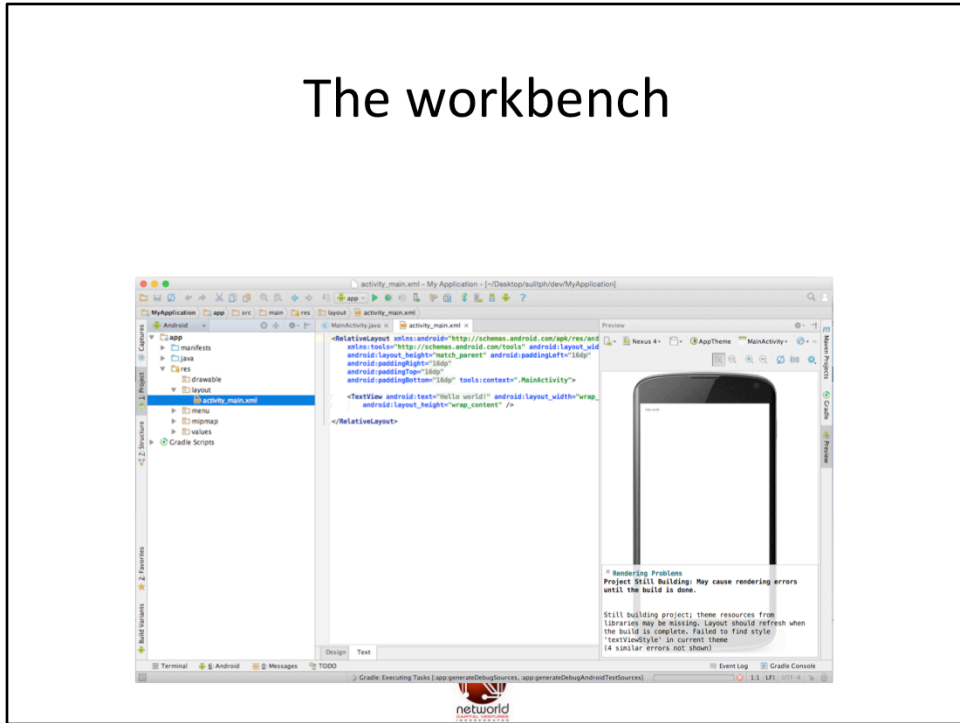


An activity is the basic program module in Android. An app has at least one Activity

Name the default files

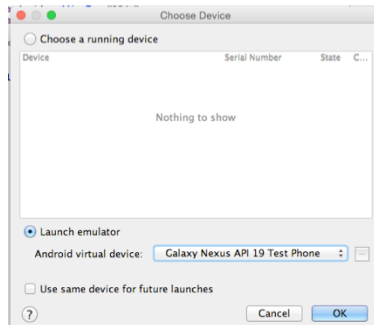


The workbench



Once gradle has finished building your new project you will be taken to the AS workbench

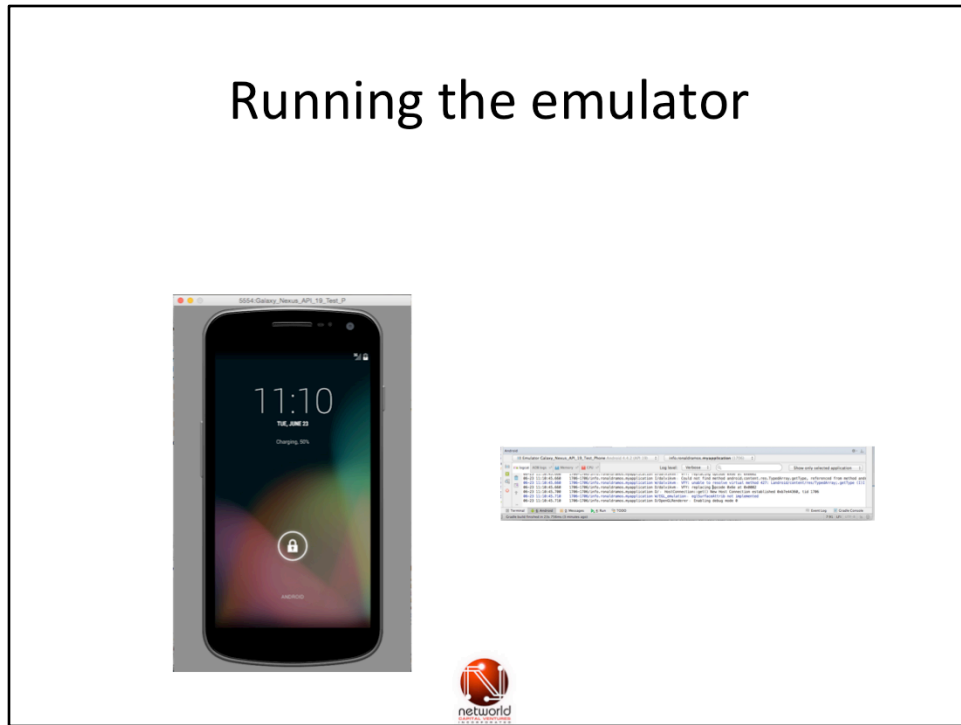
Running the project



- Click “Play” on the toolbar
- You will be asked where to run your app.
- Choose “Launch Emulator”



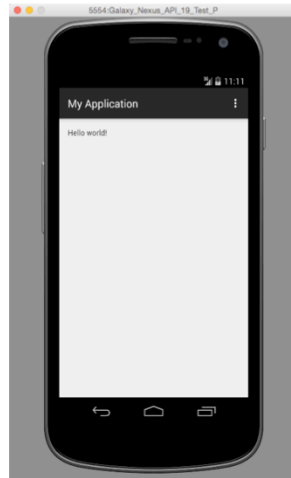
Running the emulator



The console on Android Studio will display status messages regarding the running app.

The emulator should also boot up on your PC

The running app



YOUR FIRST ANDROID APP



Create a New Android Project

- Click on File > New > Project
- Select Android Application



Create a new Android Project

- Project Name:
 - MyFirstForm
- Min SDK: Android 4.0
- Application Name:
 - My First Form
- Package Name
 - com.yourname.myfi
rstproject
- Activity Name:
 - MyFirstActivity



Create a new android project in
Android Studio

Create a New Android Project

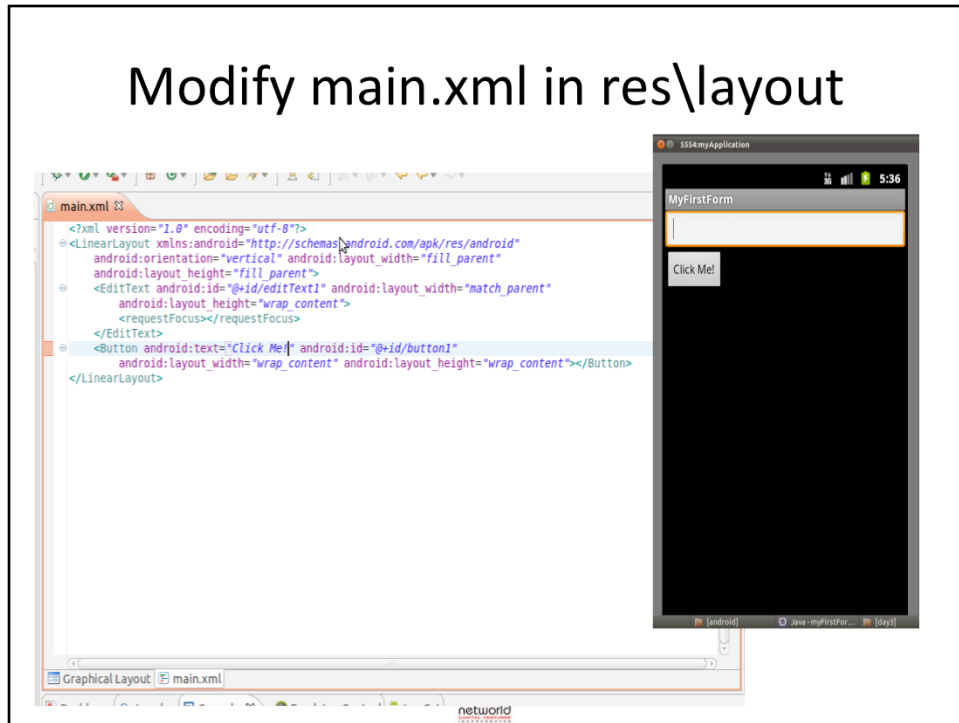
- Click on Finish
- Once the project appears in project explorer you have just completed your first android app
- Try it by clicking run



MODIFYING YOUR PROJECT



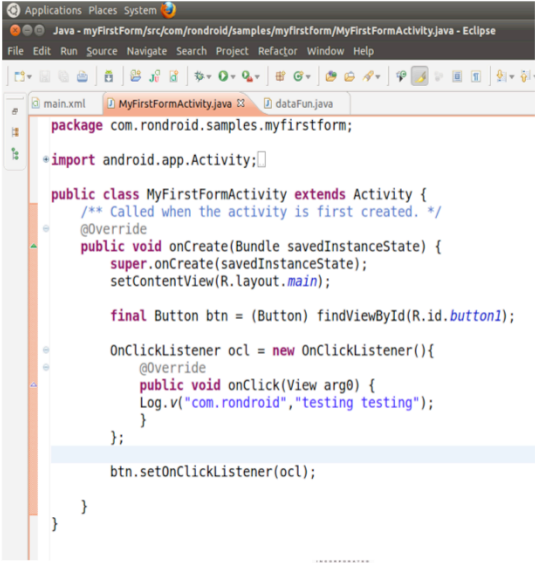
Modify main.xml in res\layout



Modify the layout main.xml in the res
\layout folder

It will contain one EditText (a textbox)
and one Button

Modify the Activity



```
package com.rondroid.samples.myfirstform;

import android.app.Activity;

public class MyFirstFormActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btn = (Button) findViewById(R.id.button1);

        OnClickListener ocl = new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                Log.v("com.rondroid", "testing testing");
            }
        };

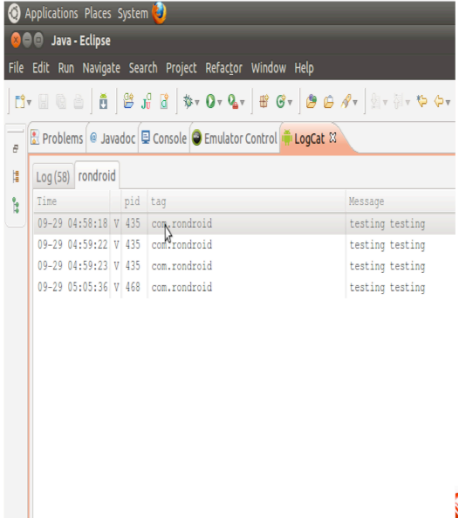
        btn.setOnClickListener(ocl);
    }
}
```

Code the activity.

The activity will do the following:

1. Add an event listener (onClick) to the button
2. When someone clicks an entry will be posted to Android's built in log


Run the Activity



The screenshot shows the Eclipse IDE interface. The top menu bar includes 'File', 'Edit', 'Run', 'Navigate', 'Search', 'Project', 'Refactor', 'Window', and 'Help'. The toolbar contains various icons for file operations and development tools. The 'LogCat' window is open, displaying a table of log entries for the package 'com.rondroid'. The table has columns for 'Time', 'pid', 'tag', and 'Message'. The log entries are as follows:

Time	pid	tag	Message
09-29 04:58:18	V 435	com.rondroid	testing testing
09-29 04:59:22	V 435	com.rondroid	testing testing
09-29 04:59:23	V 435	com.rondroid	testing testing
09-29 05:05:36	V 468	com.rondroid	testing testing

- Make sure you have the LogCat screen



To see the result add a log filter to Eclipse's logcat window

Everytime you click on the button it will reflect in logcat

Modify the Activity again

```
package com.rondroid.samples.myfirstform;

import android.app.Activity;

public class MyFirstFormActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btn = (Button) findViewById(R.id.button1);
        final EditText et = (EditText) findViewById(R.id.editText1);

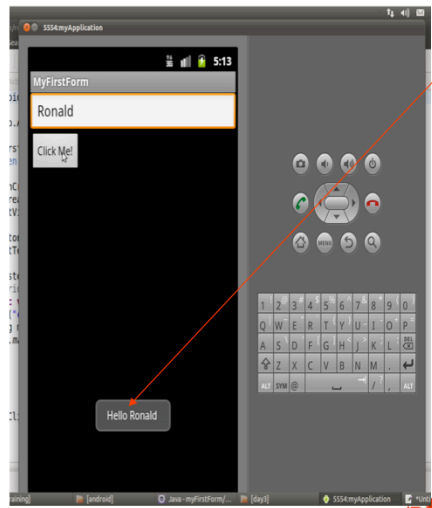
        OnClickListener ocl = new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                Log.v("com.rondroid", "testing testing");
                String msg = et.getText().toString();
                Toast.makeText(getApplicationContext(),
                    "Hello " + msg ,
                    Toast.LENGTH_LONG).show();
            }
        };

        btn.setOnClickListener(ocl);
    }
}
```

Now that we know our button is working we can add more complex code.

The code we are adding will get whatever text is in the EditText and display it in a pop-up message.

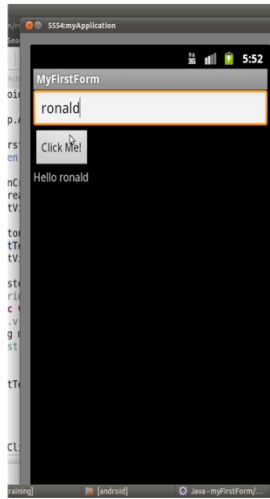
2nd run of application



Toast message appears on the screen reflecting value in EditText control



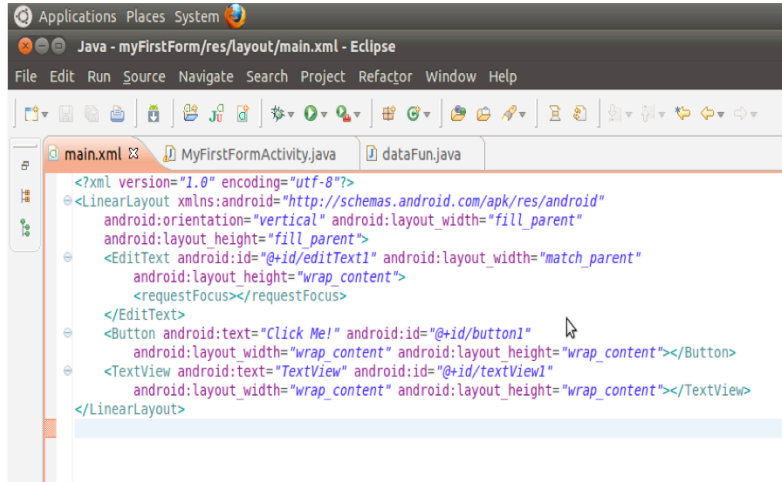
Example:



- Create the following form
 - It contains an EditText, Button and TextView
- Whenever you click the button it will display "Hello " + the content of the EditText in the TextView



Main.xml



The screenshot shows the Eclipse IDE interface with the following XML code in the main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:id="@+id/editText1" android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <requestFocus></requestFocus>
    </EditText>
    <Button android:text="Click Me!" android:id="@+id/button1"
        android:layout_width="wrap_content" android:layout_height="wrap_content"></Button>
    <TextView android:text="TextView" android:id="@+id/textView1"
        android:layout_width="wrap_content" android:layout_height="wrap_content"></TextView>
</LinearLayout>
```



Activity code

```
public class MyFirstFormActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btn = (Button) findViewById(R.id.button1);
        final EditText et = (EditText) findViewById(R.id.editText1);
        final TextView tv = (TextView) findViewById(R.id.textView1);

        OnClickListener ocl = new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                //Log.v("com.rondroid", "testing testing");
                String msg = et.getText().toString();
                //Toast.makeText(getApplicationContext(),
                //    "Hello " + msg ,
                //    Toast.LENGTH_LONG).show();
                tv.setText("Hello " + msg);
            }
        };

        btn.setOnClickListener(ocl);
    }
}
```



PRESENTING DATA: LISTVIEWS

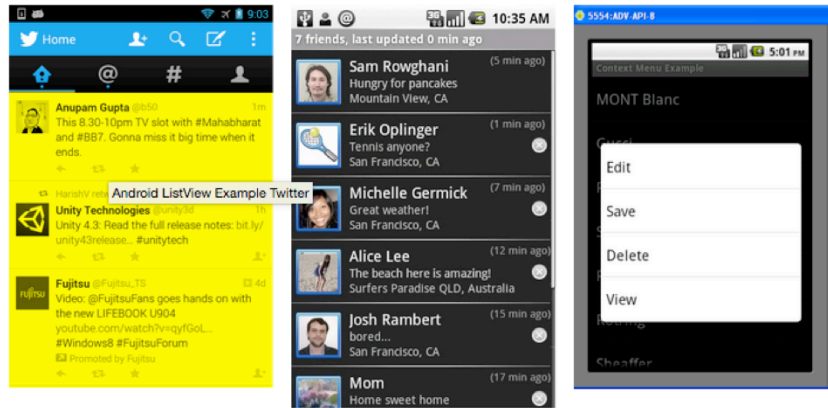


What is a ListView?

- Android provides the view "ListView" which is capable of displaying a scrollable list of items.
 - "ListView" gets the data to display via an adapter.
 - An adapter which must extend "BaseAdapter" and is responsible for providing the data model for the list and for converting the data into the fields of the list.
- Android has two standard adapters, ArrayAdapter and CursorAdapter .
 - "ArrayAdapter" can handle data based on Arrays or Lists
 - "SimpleCursorAdapter" handle database related data.
- You can develop your own Adapter by extending these classes or the BaseAdapter class.



ListView Examples



<http://www.codelearn.org/android-tutorial/android-listview>

ListItem

- An Android listview is made from a group of list items.
- List items are individual rows in listview where the data will be displayed.
- Any data in listview is displayed only through listItem. Consider Listview as scrollable group of list items.



An Android listview is made from a group of list items. List items are individual rows in listview where the data will be displayed. Any data in listview is displayed only through listItem. Consider Listview as scrollable group of list items.

List items are just layouts in a separate layout file. Let us understand the following example.

Android Listview Twitter tweet example

Here we can see a listItem for the twitter application. This list Item is arranged in a Relative layout with images and multiple text views aligned to each other. This is how an Android listview is designed.

Once we have the listItem, we bind the listview to the Adapter and then use list items to display the data in listview.

Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <ListView android:id="@+id/ListView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```



Activity code

```
public class MyListViewActivity extends Activity {
    /** Called when the activity is first created. */
    private ListView lvl;
    private String
lv_arr[]={ "Android", "iPhone", "BlackBerry", "AndroidPeople" };
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        lvl=(ListView)findViewById(R.id.ListView01);
        // By using setAdapter method in listview we can add string array in
list.
        lvl.setAdapter(new
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1 , lv_arr));
    }
}
```



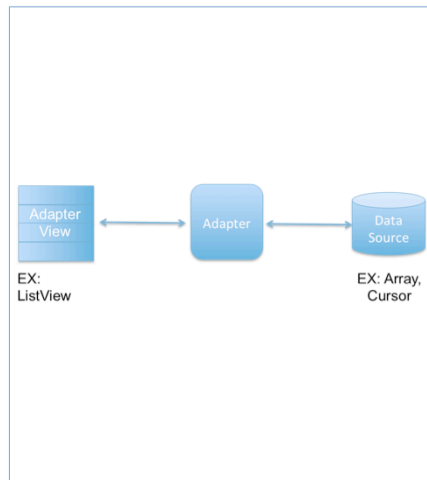
Adding Click Reactions

```
private ListView lvl;
private String lv_arr[]{"Android","iPhone","BlackBerry","AndroidPeople"};
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    lvl=(ListView)findViewById(R.id.ListView01);
    lvl.setOnItemClickListener(this);
    // By using setAdpater method in listview we an add string array in list.
    lvl.setAdapter(new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1 , lv_arr));
}

public void onItemClick(AdapterView arg0, View v, int position, long arg3) {
    // TODO Auto-generated method stub
    Toast.makeText(this, "u clicked " + lv_arr[position] ,Toast.LENGTH_LONG).show();
}
```



Adapters



- Android Adapter is a bridge between the View (e.g. ListView) and the underlying data for that view. An adapter manages the data and adapts the data to the individual rows (listItems) of the view.
- We bind the adapter with Android listview via setAdapter method.



Adapter

Android Adapter is a bridge between the View (e.g. ListView) and the underlying data for that view. An adapter manages the data and adapts the data to the individual rows (listItems) of the view.

We bind the adapter with Android listview via setAdapter method. Now, Let us see how adapter works with the help of the following image.

Android Listview Adapters

As stated earlier, Adapters act as a bridge to the views. To interact with the view, adapters call the getView() method which returns a view for each item within the adapter view. This is a listitem which we have seen earlier. The layout format and the corresponding data for an item within the adapter view are set in the getView() method.

Once we have a reference to the view, we can get the data from the data source either in an Array list or a cursor. We can then bind the data to the view items. All this is done in getView Method. In coming sections, we will look as to how we can achieve this in our code.



Android's built-in DB

SQLITE DATABASES AND ANDROID



SQLite

- SQLite is an Open Source Database which is embedded into Android. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. In addition it requires only little memory at runtime (approx. 250 KByte).
- Using SQLite in Android does not require any database setup or administration. You specify the SQL for working with the database and the database is automatically managed for you.



Working with databases in Android can be slow due to the necessary I/O. Therefore it is recommended to perform this task in an AsyncTask .

SQLite supports the data types TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before saving them in the database. SQLite itself does not validate if the types written to the columns are actually of the defined type, you can write an integer into a string column.

If your application creates a database this database is saved in the directory "DATA/data/APP_NAME/databases/FILENAME". "DATA" is the path which Environment.getDataDirectory() returns, "APP_NAME" is your application name and "FILENAME" is the name you give the database during creation. Environment.getDataDirectory() usually returns the SD card as location.

A SQLite database is private to the application which creates it. If you want to share data with other applications you can use a Content Provider.

SQLiteOpenHelper

- To create and upgrade a database in your Android application you usually subclass "SQLiteOpenHelper". In this class you need to override the methods onCreate() to create the database and onUpgrade() to upgrade the database in case of changes in the database schema. Both methods receive an "SQLiteDatabase" object.
- SQLiteOpenHelper provides the methods getReadableDatabase() and getWritableDatabase() to get access to an "SQLiteDatabase" object which allows database access either in read or write mode.
- For the primary key of the database you should always use the identifier "_id" as some of Android functions rely on this standard.



SQLiteDatabase methods

- "SQLiteDatabase" provides the methods insert(), update() and delete() and execSQL() method which allows to execute SQL directly.
- The object "ContentValues" allow to define key/values for insert and update. The key is the column and the value is the value for this column.
- Queries can be created via the method.rawQuery() which accepts SQL or query() which provides an interface for specifying dynamic data or SQLiteQueryBuilder.



SQLiteBuilder is similar to the interface of an content provider therefore it is typically used in ContentProviders. A query always returns a "Cursor".

The method query has the parameters String dbName, int[] columnNames, String whereClause, String[] valuesForWhereClause, String[] groupBy, String[] having, String[] orderBy. If all data should be selected you can pass "null" as the where clause. The where clause is specified without "where", for example "_id=19 and summary=?". If several values are required via ? you pass them in the valuesForWhereClause array to the query. In general if something is not required you can pass "null", e.g. for the group by clause.

Cursor

- A Cursor represents the result of a query. To get the number of elements use the method `getCount()`. To move between individual data rows, you can use the methods `moveToFirst()` and `moveToNext()`. Via the method `isAfterLast()` you can check if there is still some data.
- A Cursor can be directly used via the "SimpleCursorAdapter" in ListViews.

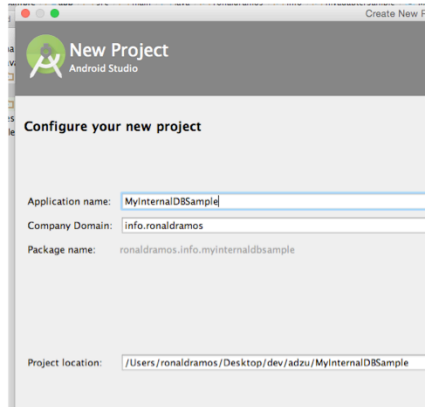


Getting your hands dirty...

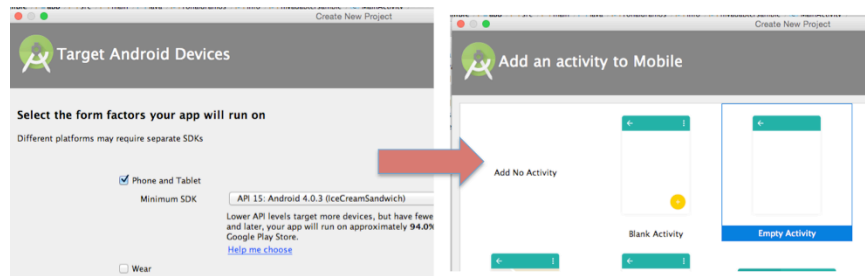
A SAMPLE DB APP



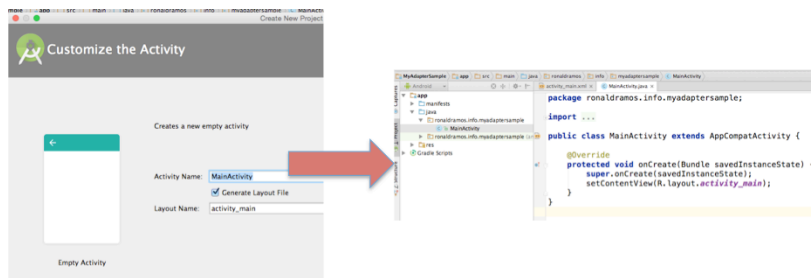
Create a new project



Create a new project



Create a new project



How the app components are split

- **App layer**
 - UI
- **Data storage layer**
 - application specific and deals with creating the database and operations on it but these are still database independent.
- **SQLite layer**
 - deals with DB access
- **Specific operations through a db helper**



Systems Design

While the app is very simple the plumbing around constructing a database is complex at first. The layered architecture is shown in the figure below. The operation on the SQLite database is wrapped in a few layers of abstraction. I view these layers as follows:

- app layer -- just deals with comments and the UI
- data storage layer -- application specific and deals with creating the database and operations on it but these are still database independent.
- SQLite layer -- this deals with all the plumbing and database specific operations through a db helper

You won't see these layers necessarily like this in the literature but I think it helps explain the architecture and operations. There is a separation of concerns: the app deals with objects that matter to it, i.e., comments; and the lower layers deal with storage of data, and eventually the detail operations of an SQL database. You want to hide these details from the user. For example, I'm sure you are thinking -- why make this so hard? Why not have the app reach down into the SQLite code and just leave it like that. But we use abstraction -- of layering -- to hide the grubby details from the app layer.

Person class

```
package ronaldramos.info.myinternaldbsample;
/**
 * Class to hold a single Person (1 record)
 */
public class Person {
    private long id;
    private String name;

    public void setId(long id){
        this.id = id;
    }

    public long getId(){
        return this.id;
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return this.name;
    }

    // Will be used by the ArrayAdapter in the ListView
    @Override
    public String toString() {
        return name;
    }
}
```



This class will represent one record in the table.

The application simply allows the user to add names to the database and display them to the UI. Names can be added one at a time, deleted (from the top) one at a time or all names can be deleted. So *Person* is the main object that the user deals with. The Person class includes getters/setters for id and name. Every time a person needs to be added a Person is instantiated. The Person class is the app model and contains the data that is inserted, queried and deleted in the database. Names are also shown in the UI.

CODE:

```
package ronaldramos.info.myinternaldbsample;
/**
 * Class to hold a single Person (1 record)
 */
public class Person {
    private long id;
    private String name;

    public void setId(long id){
        this.id = id;
    }
}
```

SQLiteHelper class

```
package ronaldramos.info.myinternaldbsample;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
/**Class to facilitate creation and upgrade of DB*/
public class MySQLiteHelper extends SQLiteOpenHelper {
    public static final String TABLE_NAME = "persons";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_NAME = "name";
    private static final String DATABASE_NAME = "personsDB";
    private static final int DATABASE_VERSION = 1;
    // Database creation sql statement
    private static final String DATABASE_CREATE = "create table "
        + TABLE_NAME + "(" + COLUMN_ID + " integer primary key autoincrement, "
        + COLUMN_NAME + " text not null)";
    public MySQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(MySQLiteHelper.class.getName(), "Upgrading database from version "
            + oldVersion + " to " + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```



The MySQLiteHelper (which extends SQLiteOpenHelper) implements the helper class to manage database creation and version management. It implements onCreate() and onUpgrade() to take care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state (e.g., database.execSQL(DATABASE_CREATE)).

Code:

```
package ronaldramos.info.myinternaldbsample;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
/**Class to facilitate creation and upgrade of DB*/
public class MySQLiteHelper extends SQLiteOpenHelper {
    public static final String TABLE_NAME = "persons";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_NAME = "name";
    private static final String DATABASE_NAME = "personsDB";
    private static final int DATABASE_VERSION = 1;
    // Database creation sql statement
    private static final String DATABASE_CREATE = "create table "
```

The Data Storage Layer: DBHelper class

```
package ronaldramos.info.myinternaldbsample;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;

import java.sql.SQLException;
/**Class to facilitate data manipulation.*/
public class DBHelper {
    // Database fields
    private SQLiteDatabase database;
    private MySQLiteHelper SQLHelper;
    private String[] allColumns = { MySQLiteHelper.COLUMN_ID,
        MySQLiteHelper.COLUMN_NAME };

    private static final String TAG = "DBDEMO";

    public DBHelper(Context context) {
        SQLHelper = new MySQLiteHelper(context);
    }

    public void open() throws SQLException {
        database = SQLHelper.getWritableDatabase();
    }

    public void close() {
        SQLHelper.close();
    }
}
```



DBHelper maintains the database connection and supports adding, fetching and deleting comments. CommentsDataSource creates the MySQLiteHelper class which details with the actual SQLite database. Very little of the internal details of the SQLite layer are exposed to the data storage layer, as you can see in the code below. The database constants such as the column names are exposed; in our case just one column name: COLUMN_COMMENT and the column ID: COLUMN_ID. These database constants are made public by the SQLite layer discussed in the next section. These public constants are needed for inserting and querying comment objects in the database.

The constructor creates the database. It first creates a db helper MySQLiteOpenHelper to deal with operations on the database. After the db helper is created the constructor opens the database, as shown the code below.

The database is a SQLiteDatabase object, exposing methods to manage a SQLite database. SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.

The code calls getWritableDatabase() to open and obtain a writable instance of the underlying database using the db helper implemented in the SQLite layer. If the database does not exist the helper executes its onCreate() handler -- see

App specific operations on storage: insert and deleting

- The data storage layer also provides the key operations on the database in terms of the application specific data object, in our case: Person. These operations are:
 1. createPerson(String person), which inserts a Person in the database as content values. This method also issues a query to read back what was written.
 2. deletePerson(Person person), which deletes a Person
 3. deleteAllPersons(), empties the database.
 4. getAllPersons() , which gets all the persons listed in the DB



DBHelper – Create a Person

```
//DB data operations
public Person createPerson(String name) {
    ContentValues values = new ContentValues();
    values.put(MySQLiteHelper.COLUMN_NAME, name);
    long insertId = database.insert(MySQLiteHelper.TABLE_NAME, null,
        values);
    Cursor cursor = database.query(MySQLiteHelper.TABLE_NAME,
        allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null,
        null, null, null);
    cursor.moveToFirst();
    Person newPerson = cursorToPerson(cursor);

    // Log the comment stored
    Log.d(TAG, "comment = " + cursorToPerson(cursor).toString()
        + " insert ID = " + insertId);

    cursor.close();
    return newPerson;
}
```



CODE:

```
public Person createPerson(String name) {
    ContentValues values = new ContentValues();
    values.put(MySQLiteHelper.COLUMN_NAME, name);
    long insertId = database.insert(MySQLiteHelper.TABLE_NAME, null,
        values);
    Cursor cursor = database.query(MySQLiteHelper.TABLE_NAME,
        allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null,
        null, null, null);
    cursor.moveToFirst();
    Person newPerson = cursorToPerson(cursor);

    // Log the comment stored
    Log.d(TAG, "comment = " + cursorToPerson(cursor).toString()
        + " insert ID = " + insertId);

    cursor.close();
    return newPerson;
}
```

DBHelper – Delete records

```
public void deletePerson(Person person) {
    long id = person.getId();
    Log.d(TAG, "delete comment = " + id);
    System.out.println("Person deleted with id: " + id);
    database.delete(MySQLiteHelper.TABLE_NAME, MySQLiteHelper.COLUMN_ID
        + " = " + id, null);
}

public void deleteAllPersons() {
    System.out.println("All persons deleted all");
    Log.d(TAG, "delete all = ");
    database.delete(MySQLiteHelper.TABLE_NAME, null, null);
}
```



CODE:

```
public void deletePerson(Person person) {
    long id = person.getId();
    Log.d(TAG, "delete comment = " + id);
    System.out.println("Person deleted with id: " + id);
    database.delete(MySQLiteHelper.TABLE_NAME, MySQLiteHelper.COLUMN_ID
        + " = " + id, null);
}

public void deleteAllPersons() {
    System.out.println("All persons deleted all");
    Log.d(TAG, "delete all = ");
    database.delete(MySQLiteHelper.TABLE_NAME, null, null);
}
```

DBHelper – Other functions

```
public List<Person> getAllComments() {
    List<Person> persons = new ArrayList<Person>();

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

    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        Person person = cursorToPerson(cursor);
        Log.d(TAG, "get comment = " + cursorToPerson(cursor).toString());
        persons.add(person);
        cursor.moveToNext();
    }
    // Make sure to close the cursor
    cursor.close();
    return persons;
}

private Person cursorToPerson(Cursor cursor) {
    Person person = new Person();
    person.setId(cursor.getLong(0));
    person.setName(cursor.getString(1));
    return person;
}
```



CODE:

```
public List<Person> getAllComments() {
    List<Person> persons = new ArrayList<Person>();

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

    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        Person person = cursorToPerson(cursor);
        Log.d(TAG, "get comment = " + cursorToPerson(cursor).toString());
        persons.add(person);
        cursor.moveToNext();
    }
    // Make sure to close the cursor
    cursor.close();
    return persons;
}
```

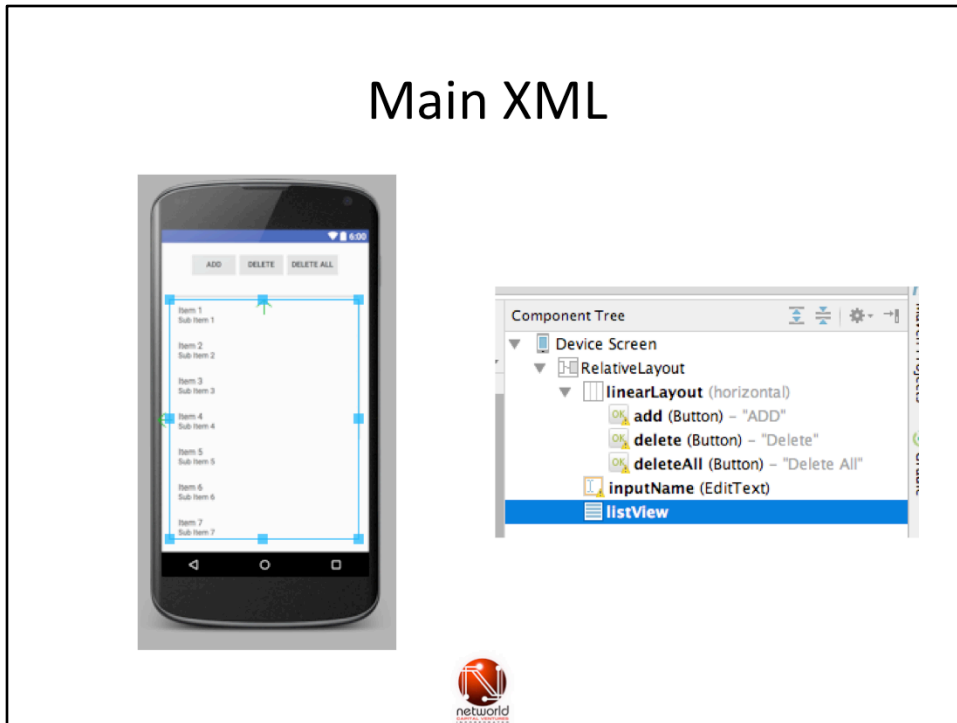
USING THE HELPER CLASSES



The App



Main XML



CODE:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/
activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:id="@+id/linearLayout"
    android:gravity="center">

<Button
```

MainActivity

onCreate()

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

    datasource = new DBHelper(this);
    try{
        datasource.open();
    }
    catch (Exception e){

    }
    ListView lv = (ListView)findViewById(R.id.ListView);

    List<Person> values = datasource.getAllComments();

    // Use the SimpleCursorAdapter to show the
    // elements in a ListView
    ArrayAdapter<Person> adapter = new ArrayAdapter<Person>(this,
        android.R.layout.simple_list_item_1, values);
    lv.setAdapter(adapter);
}
```

Process()

```
public void process(View view) {
    @SuppressWarnings("unchecked")
    ListView lv = (ListView)findViewById(R.id.ListView);
    ArrayAdapter<Person> adapter = (ArrayAdapter<Person>) lv.getAdapter();
    Person person = null;
    EditText et = (EditText)findViewById(R.id.inputName);
    switch (view.getId()) {
        case R.id.add:
            String name = et.getText().toString();
            // Save the new comment to the database
            person = datasource.createPerson(name);
            adapter.add(person);
            break;
        case R.id.delete:
            if (lv.getAdapter().getCount() > 0) {
                person = (Person) lv.getAdapter().getItem(0);
                datasource.deletePerson(person);
                adapter.remove(person);
            }
            break;
        case R.id.deleteAll:
            if (lv.getAdapter().getCount() > 0) {
                datasource.deleteAllPersons();
                adapter.clear();
            }
            break;
    }
    adapter.notifyDataSetChanged();
}
```



CODE

```
package ronaldramos.info.myinternaldbsample;
```

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
```

```
import java.sql.SQLException;
import java.util.List;
import java.util.Random;
```

```
public class MainActivity extends AppCompatActivity{
    private DBHelper datasource;
```

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


END DAY 1

