

Introduction to Android Programming

by

Qaiser Siddique

Helsinki Metropolia University of Applied Science

Preface

This is hands on lab course and this document is intended for developers who wants to get started with android application development. In this document overview of different components of an android application is discussed briefly. Most of the code to program an application (Movies Database) is given in this material. Some part is left for the developers to program in order to have a complete application in the end.

Movies Database App will be created in this document, which will have at least following features

- Adding Movie via user input
- Searching for Movie using web service
- Movie details view
- Saving the Movie to database
- Viewing list of movies in the database
- Deleting a movie from database

Note: This application is designed for learning purposes only, there are several aspects not considered like data validation, activity life cycle managements and others. To get full understanding of the topic developer must read the reference documentation on android.com.

Table of Contents

Preface	1
1. Setting up the Environment	4
1.1 Installation on your computer	4
1.2 Starting on Metropolia Labs	4
2. Creating and Running Android Project	4
2.1 Creating new project	4
2.2 Creating and running in Emulator	7
2.3 Running on Android Device	8
3. View and Controller	9
3.1 Locating Controller and View files	9
3.2 Editing View File	9
3.3 Event Handling	12
3.3.1 Activity Life Cycle Methods	13
3.4 Adding new Activity	14
3.4.1 Adding Controller	14
3.4.2 Adding View	14
3.4.3 Connecting Controller and View	14
4. Intents and Data Sharing	15
4.1 Navigation using Intents	15
4.2 Passing data	16
4.2.1 Creating Input Fields	16
4.2.2 Getting and Sending User Data	17
4.2.3 Retrieve data from Intent and display	19
5. Web Services	21
5.1 Permission	22
5.2 API Call and Response	22
5.3 Parsing JSON and Custom Objects	26
6. Data Persistence	30
6.1 SQLite	30

6.1.1	SQLite Writing	32
6.1.2	SQLite Reading	33
7.	ListViews and custom Listitems	34
7.1	Creating ListView	35
7.1	Array Adapters	36
7.2	Custom Array Adapter	36
7.3	List item Click Listener.....	39
8.	Dialogue Alert	40
9.	Google Maps	41
9.1	Setting up Google Play Services	41
9.2	Displaying the debug certificate fingerprint.....	42
9.3	Obtaining an API Key	42
9.4	Add API Key to your application	43
9.5	Setting up Application and Manifest	43
9.6	Adding the Map to your Application	45
9.7	Using Google Map Class	45
10.	Geocoding.....	46
11.	Location Services (Optional).....	47
12.	Menus.....	51
13.	Supporting Multiple Screen Sizes (Android.com, 2014)	53
13.1	Create Different Layouts	53
13.2	Create Different Bitmaps.....	54
14.	Important Links.....	56
14.1	Android Training	56
14.2	User Interfaces	56
14.3	Best practices	56
14.4	Download default icons	56
14.5	Device art generator	57
14.6	Creating icons for different screens (Assets studio)	57
14.7	Publishing check list	57
Lab 1 Extras Better UI :	58
References		59

1. Setting up the Environment

1.1 Installation on your computer

1. Go to <https://developer.android.com/sdk/index.html> and download Eclipse ADT See figure 1.
2. With a single download, the Eclipse ADT bundle includes everything you need to begin developing apps:
 - Eclipse + ADT plugin
 - Android SDK Tools
 - Android Platform-tools
 - A version of the Android platform
 - A version of the Android system image for the emulator
3. Unzip the downloaded file and Goto Eclipse Folder and run Eclipse.exe (That's It if you have java installed)



Figure 1 Downloading Android SDK

1.2 Starting on Metropolia Labs

1. Go to VMWare
2. Connect to Android Lab
3. Run Eclipse

2. Creating and Running Android Project

2.1 Creating new project

1. In Eclipse go to File > New > Android Application project as shown in figure 2

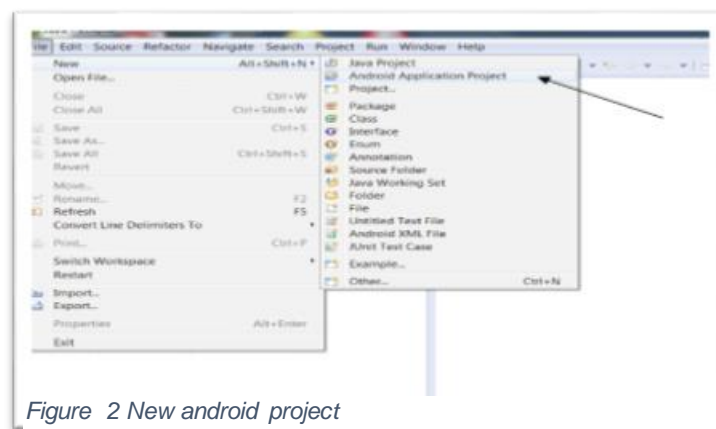


Figure 2 New android project

2. We have to insert the application and project name (Figure 3). Also the package name has to be indicated. Please use your combination of your firstname and lastname when writing the package name. We select the SDK level that we will use in our application, In this course we will work with API 19 and support the application to for API 8 which is Android 2.2 mobile device. These details can also be changed in the Manifest File later on.

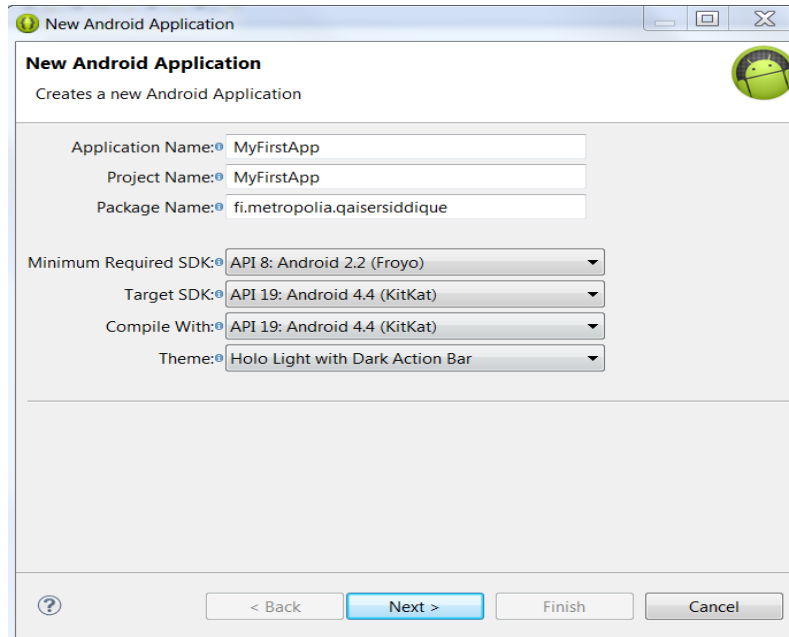


Figure 3 Naming the project

3. In the next step (figure 4) we check the Custom Launcher Icon and Create our first Activity which means our first screen will automatically be created and we will have Hello World example to start with.

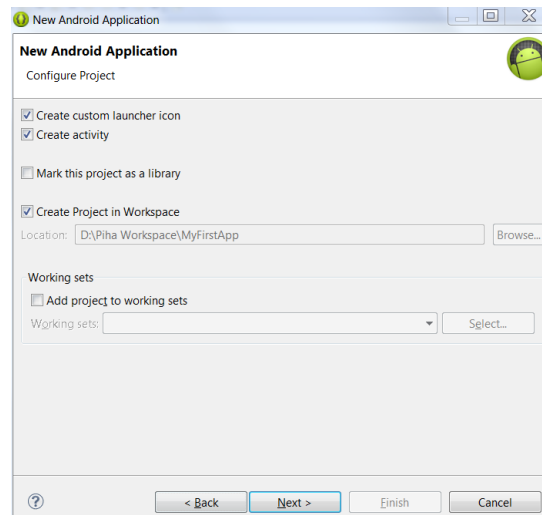


Figure 4 Project configurations

4. In the next screen we can choose the application icon (figure 5). By default, an icon with different resolutions is shown, but we can change it by loading another icon from an external file (e.g., a png file). The icon can be changed later. Android Assets Studio (online tool) can also be used to create different sizes of one launcher icon.

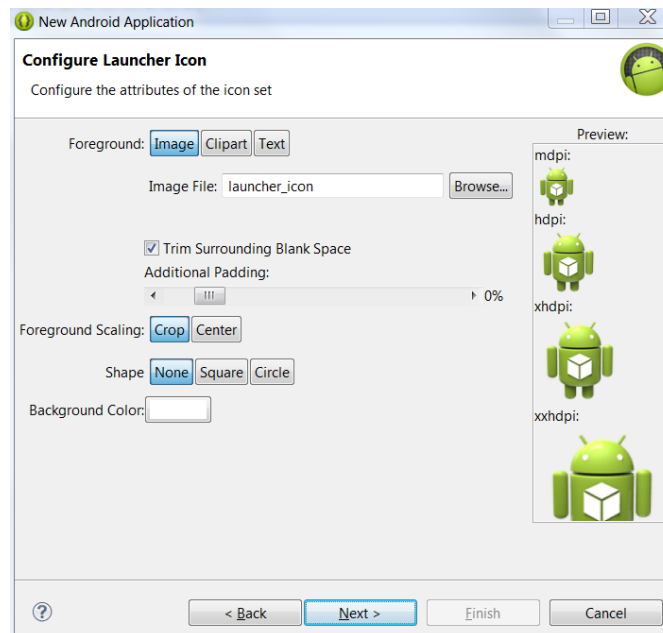


Figure 5 Application icons

5. We will start our project with Black Activity, however other templates are also available which will auto-generate most of the code like for Navigation drawer, Master/Detail flow, Tabbed activity etc.
6. Name your activity and its layout file. We can keep it as MainActivity. Then click finish button.

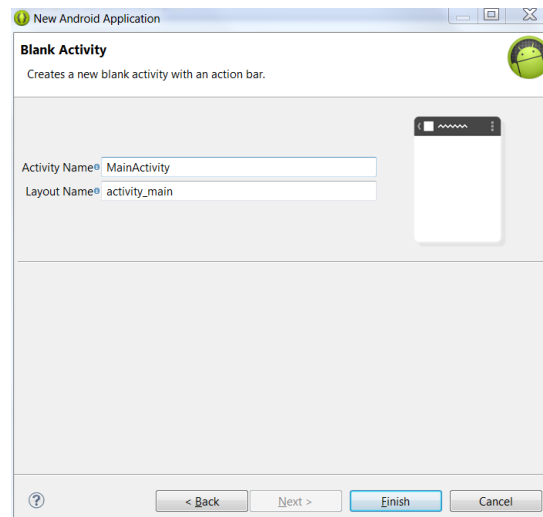


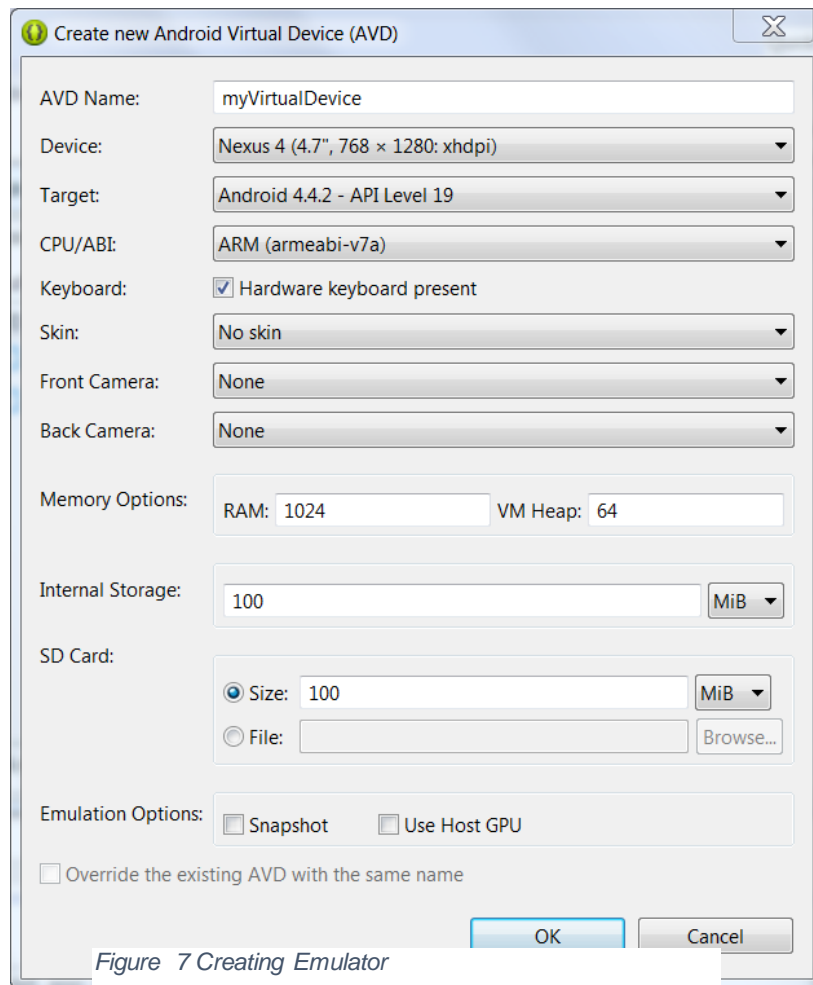
Figure 6 Default activity

2.2 Creating and running in Emulator

Once project is successfully created, by default “Hello World!” view is created. Now we will create an emulator and run the application on emulator.

Important Note: Emulator takes long time to start, once it's started do not close it, every time you make changes to the code, App is updated on the same instance of emulator.

1. In Eclipse menu, go to Window -> Android Virtual Device Manager
2. Click Create and set the following configurations as shown in figure 7 and click OK



3. Now you will be able to see your newly created device in the list of Virtual Device. Select your virtual device and click Start button.
4. Now from project explorer select your project and go to Run in Eclipse menu and click RUN. (if prompted select Run as Android Application Project)
5. Your application will now open up in the emulator.

Note: If emulator does not work, reduce RAM to 768 MB

2.3 Running on Android Device

If you have a real Android-powered device, here's how you can install and run your app: Plug in your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the [OEM USB Drivers](#) document.

Enable USB debugging on your device. On most devices running Android 3.2 or older, you can find the option **under Settings > Applications > Development**. On Android 4.0 and newer, it's in **Settings > Developer options**.

Note: On Android 4.2 and newer, Developer options is hidden by default. To make it available, go to **Settings > About** phone and tap Build number seven times. Return to the previous screen to find Developer options.

To run the app from Eclipse: Open one of your project's files and click **Run** from the toolbar. In the Run as window that appears, select Android Application and click **OK**. Eclipse installs the app on your connected device and starts it.

3. View and Controller

3.1 Locating Controller and View files

1. In the Project Explorer you will find your MainActivity.java file which was created during project setup under src (source)-> package name (fi.metropolia.qaisersiddique) -> MainActivity.java. see figure 8
2. activity_main.xml file will be under res (Resources) -> layout -> activity_main.xml. see figure 8
3. Double click to open activity_main.xml

3.2 Editing View File

1. There are two views available for layout file Graphical Layout and other one is xml view. Switch to xml view by clicking the tab at the bottom of Palette Panel as shown in figure 9.
2. We can see the code for Hello World text that is appearing on the screen in xml format.

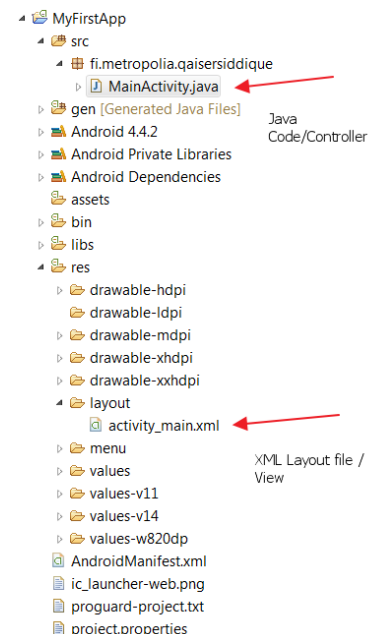


Figure 8 Location Java and XML files

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Now edit this file to create view of your MainActivity. We know our application requires many screens like Add Movie, View Movie, Search, Saved Movies etc . We will start by adding buttons to the view which will take us to next activities.

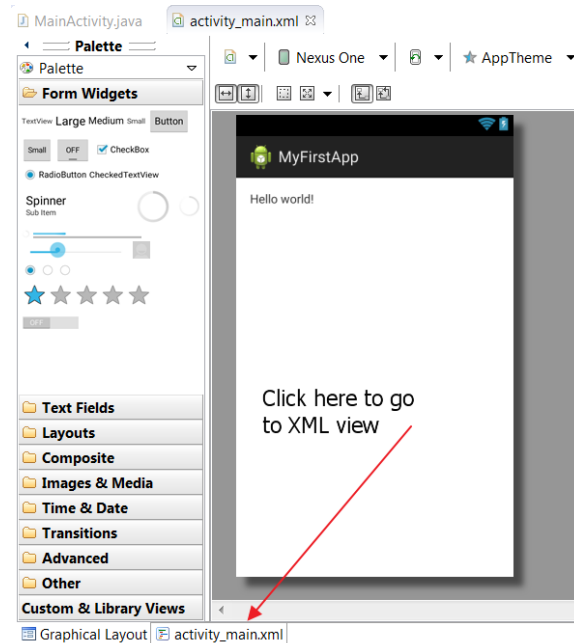


Figure 9 Layout editor

3. Now we will add a button and name it “Add Movies” . Remove the code for TextView of with Hello World and add following code to the xml file.

```
<Button
    android:id="@+id/add_movie_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:text="@string/addmovie" />
```

This will give an error “No resource found that matches the given name (at 'text' with value '@string/addmovie').” as shown in figure 10. We will resolve this error in step 4.

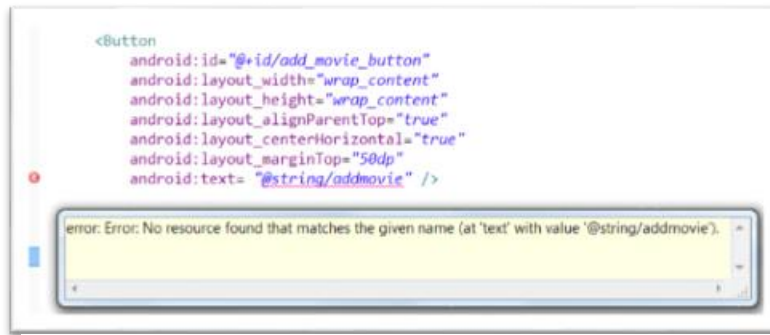


Figure 10 Errors on XML editor

4. If we replace

```
android:text= "@string/addmovie"
```

with this

```
android:text= "Add Movie"
```

than error will go away but we do not want to hard code any labels. Infact we will add a resource in Strings.xml file with the name "addmovie" and value "Add Movie" in step 5. Click [here](#) to learn more.

5. Go to res -> values -> Strings.xml then select xml format and add following line of code, save all files to resolve the error.

```
<string name="addmovie">Add Movie</string>
```

6. Similarly create other buttons in activity_main.xml with their labels in Strings.xml files as shown in the figure below. Read steps before starting to work on this User Interface.

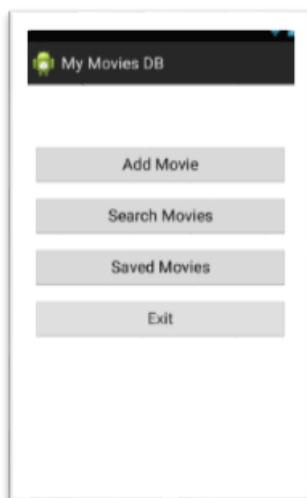


Figure 11 Main Activity layout

HINT: In order to align buttons you can use `layout_below` property. e.g when aligning Search Movies button below Add Movies button you can add following line to Search Movies button

```
android:layout_below="@+id/add_movie_button"
```

To adjust the width of the button try using “`match_parent`.”

3.3 Event Handling

Once your buttons are ready and each button has a unique ID than go to `MainActivity.java` and perform following steps.

1. Create the Button object in scope of class.

```
Button addButton;
```

2. Initialize and add reference to corresponding Button widget in the view file in `onCreate` Method.

```
addButton = (Button) findViewById(R.id.add_movie_button);
```

3. Add `onClick`Listener in `onStart` Method.

```
@Override
protected void onStart() {
    super.onStart();
    addButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // This is where we define what happens when button is clicked.
        }
    });
}
```

4. To test our click listeners we will now add a Toast Message which will appear on the screen when button is clicked. To do so we will write one method that can be called by any button to show the toast message. In order to have the name of the button displayed on the screen we will add a String parameter to the method to distinguish between buttons.

```
public void showButtonClickedToastMessage(String buttonName) {
    Toast.makeText(this, buttonName + " Clicked",
        Toast.LENGTH_SHORT).show();
}
```

Now call this method in the onClick Method and send the Label of button which can be done by Hard Coding (Not recommended) or referencing to String.xml in resources.

```
showButtonClickedToastMessage(getResources().getString(R.string.addmovie))  
OR (not recommended)  
showButtonClickedToastMessage("Add Movie");
```

3.3.1 Activity Life Cycle Methods

TASK: Read [Activity Life Cycle \(click \)](#) and just like onCreate and onStart methods, create other Activity lifecycle method e.g onPause, onResume add following line of the code in all of the Methods and see what happens when you run the app .

Note: Remember to Import Log using **import** android.util.Log; (Shortcut : Ctrl+Shift+O)

```
Log.d("Activity Life Cycle", "onStart Method called");
```

In Eclipse menu go to Window -> Show View -> Other then Select LogCat under Android menu. This is open up Log Cat view next to Console and show all your log message. As shown in figure below.

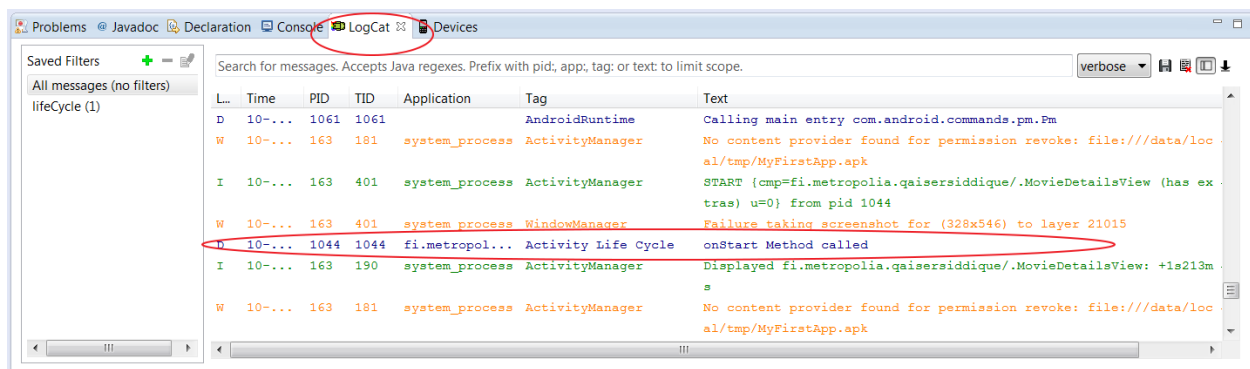


Figure 12 LogCat view

3.4 Adding new Activity

Before we can look into navigation between screens (Activities). First we will have to create new Activity and add it to application Manifest File.

3.4.1 Adding Controller

1. Right-Click on your package name (fi.metropolia.qaisersiddique) and Go to New-> Class
2. Make sure your package name is same as you set up when creating the project
3. Use Camel Case convention (<http://en.wikipedia.org/wiki/CamelCase>) and add the name of the Class. In Java Classes first letter should always be CAPITAL
e.g AddMovieActivity
4. In Superclass add : android.app.Activity , this will import and inherit the required classes for this java file to be an Android Activity
5. Click Finish
6. Now we add reference to this Class to Android Manifest file. Without the reference of all activities application crashes during the runtime.

```
<activity
    android:name=".AddMovieActivity"
    android:label="@string/addmovie" >
</activity>
```

3.4.2 Adding View

1. Go to res folder -> Layout folder. Right click on layout folder -> New -> Android XML File.
2. In the New Android XML File wizard Select resource type : Layout , check project name, add file name. layout file names are in small letters and words are separated using underscores like activity_main we can name AddMovieActivity layout to be activity_add_movie.
3. Select LinearLayout and click Finish.

Now we have a new XML Layout file where we can write the xml code for our Add Movie Activity. First we must make sure that this new layout file is connected with our Java file.

3.4.3 Connecting Controller and View

Go to controller (java file) and see if you have onCreate Method there. If it's not there then add the following code inside the class.

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

In the onCreate method we must set View of this activity to be our layout file (activity_add_movie.xml) by using setContentView command. Your onCreate should look like this.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_movie);
}
```

4. Intents and Data Sharing

4.1 Navigation using Intents

Navigation between screens of an app (activities) and parameters/data is usually transferred using Intents. There are several kinds of ready intents available for Android Developers to use. E.g if we want to open the web browser installed in the device, one can use ACTION_VIEW intent by giving it a URI

```
String url = "http://www.metropolia.fi";
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse(url));
startActivity(i);
```

similarly ACTION_VIEW can also be used to open compose mail if mailto: is used instead of URI.

Now we will look into how we can create and use our own intents. Our target is to open AddMovieActivity when Add Movie button is clicked on the MainActivity. Now in the onClick method of Add Movie Button we will create an Intent and set the destination class then start the new Activity as shown in the following code.

```
Intent in = new Intent();
in.setClass(MainActivity.this, AddMovieActivity.class);
startActivity(in);
```

Compile this code and Run it.

Note: When user presses the back button from AddMovieActivity, then application returns to MainActivity. Now see what happens when you compile and run the app again after adding following line after startActivity(in);

```
finish();
```

4.2 Passing data

4.2.1 Creating Input Fields

We can send data (strings) from one activity to another activity using Intents. Let's now create the View for Add Movie Activity and when user adds the data about the Movie then we send this data to our next activity which will be called MovieDetailsView. In your activity_add_movie xml layout file create a layout which contains atleast 3 text input fields (EditText) for the start and has atleast one button which can be DONE or ADD , choose whatever you want to name it. We will create the basic page now and in the later phase of application development you will add more fields to it. For now 3 fields must be

- Name of the Movie (or Title)
- Year when it was released
- Genre of the Movie

You should add the hint text which disappears automatically when user gets the focus on the EditText field. See figure 13 for reference

Example code for Label and Edit Text Field

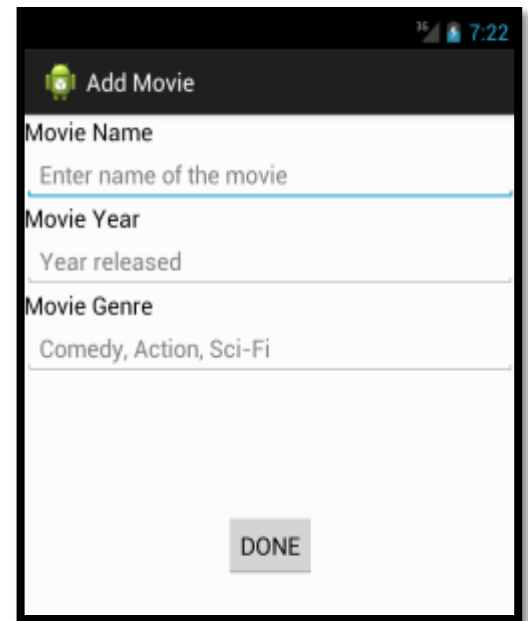


Figure 13 Add Movies Layout


```
<TextView
    android:id="@+id/TV_movieYear"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/movieyear"
    android:textAppearance="?android:attr/textAppearanceMedium"
/>

<EditText
    android:id="@+id/ET_movieYear "
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="@string/year_released"
    android:inputType="number" />
```

Note: In the Edit text widget there is a property called `inputType`, in this case we know that user will enter the year when movie was released which will be a number. If we define the `inputType` to be a number then Android Framework will automatically open the Numeric Keyboard for the user.

4.2.2 Getting and Sending User Data

To get data which user has entered we have to define Edit text box, initialize it with reference to edit text widget and then get the text that user has entered. We will save text in strings than send them to next Activity using Intent Extras.

1. Define the required variables in the scope of class

```
EditText et_movieName, et_movieYear, et_movieGenre;
String movieName, movieYear, movieGenre;
Button doneButton;
```

2. Initialize the objects in `onCreate` method of `AddMovieActivity`

```
et_movieName = (EditText)findViewById(R.id.ET_movie_name);
et_movieYear = (EditText)findViewById(R.id.ET_movie_year);
et_movieGenre = (EditText)findViewById(R.id.ET_genre);
doneButton = (Button)findViewById(R.id.btn_done);
```

3. Add a click listener to the Done Button in the onStart method, when done Button is clicked we want to validate if all the data is correct, once Data is validated then we want to send it to next Activity using Intent Extras.

Note: You must right the validation code.

```
@Override
protected void onStart() {
    // TODO Auto-generated method stub
    super.onStart();

    doneButton.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            getAndSendText();
        }
    });
}
```

4. When onClicked is called then it will refer to a method called getAndSendText() , in this method we will actually get the text which user has entered in the EditText box and send that to next activity.

```
protected void getAndSendText() {  
    // Validate the data and look for empty fields  
    movieName = et_movieName.getText().toString();  
    movieYear = et_movieYear.getText().toString();  
    movieGenre = et_movieGenre.getText().toString();  
  
    Intent in = new Intent();  
    in.setClass(AddMovieActivity.this,  
MovieDetailsView.class);  
    in.putExtra("mName", movieName);  
    in.putExtra("mYear", movieYear);  
    in.putExtra("mGenre", movieGenre);  
    startActivity(in);  
    finish();  
}
```

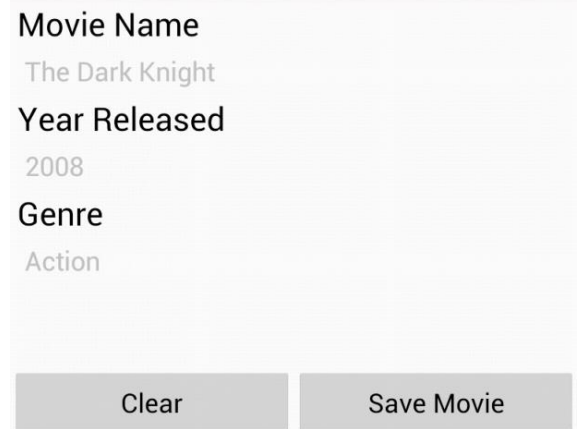
Note: In this case putExtra method of Intent takes key-value pairs and adds it to bundle which will be retrieved when we call getIntent() method in the next activity.

TASK: Validate the data before sending to the next activity. Make sure none of the input fields are empty.

4.2.3 Retrieve data from Intent and display

Now we create our next Activity called MovieDetailsView, where we will get the data from Intent and display it on the screen. In this case, we need atleast three TextView fields to display movie name, year release and genre. Create the new class and it's view. (see Section 3.4 for help). Sample view is shown in figure 14

Note: Remember to add the new Activity in manifest.



Movie Name
The Dark Knight

Year Released
2008

Genre
Action

Clear Save Movie

Figure 14 Movie Details View layout

```
mNameTV = (TextView) findViewById(R.id.tv_movie_name);  
mYearTV = (TextView)  
findViewById(R.id.tv_year_released);  
mGenreTV = (TextView) findViewById(R.id.tv_genre);
```

In the above Movie details view there are 3 TextViews where data from previous activity will be shown. We repeat the process of initializing the TextViews, referencing it in onCreate method of AddMovieActivity.

Now in the onStart() method we will get the bundle and Intent Extras and try to retrieve the values of keys we set in the last activity.

```
@Override  
protected void onStart() {  
    super.onStart();  
    Intent intent = getIntent();  
    Bundle extras = intent.getExtras();  
    if (extras != null) {  
        String mName = extras.getString("mName");  
        String mYear = extras.getString("mYear");  
        String mGenre = extras.getString("mGenre");  
        mNameTV.setText(mName);  
        mYearTV.setText(mYear);  
        mGenreTV.setText(mGenre);  
    } else {  
        mNameTV.setText("Extras was null");  
        mYearTV.setText("Extras was null");  
        mGenreTV.setText("Extras was null");  
    }  
}
```

First we get the intent then we get extras and finally we get String using the keys that was put in the previous activity. Once we have the String we are saving it in a local variable and using that variable to setText of the TextView. Alternatively we can also perform the whole action in one line as shown below, but in this approach one must be sure there are extras in the Intent.

```
mNameTV.setText(getIntent().getExtras().getString("mName"));
```

TASK:

Clear Button: When clear button is pressed, Application should kill current activity and go to Main Activity.

Note: We will create functionality of save button when we study Data Persistence later in this course.

5. Web Services

In this section we will learn how to use web services by making API call then reading the response, parsing data into custom objects.

We will use Open Movie Data Base API to make a web request and API returns JSON in response. We will parse the JSON data and create our own object of Movie which will have properties name, year released , genre and others. First we will see how OMDb API works

Go to the browser and enter this url

URL: <http://www.omdbapi.com/?t=titanic>

Response is generated in JSON format which contains movie details as shown in figure below.



```
{
  Title: "Titanic",
  Year: "1997",
  Rated: "PG-13",
  Released: "19 Dec 1997",
  Runtime: "194 min",
  Genre: "Drama, Romance",
  Director: "James Cameron",
  Writer: "James Cameron",
  Actors: "Leonardo DiCaprio, Kate Winslet, Billy Zane, Kathy Bates",
  Plot: "A seventeen-year-old aristocrat, expecting to be married to a rich claimant by her mother, falls in love with a kind but poor artist aboard the luxurious, ill-fated R.M.S. Titanic.",
  Language: "English, French, German, Swedish, Italian, Russian",
  Country: "USA",
  Awards: "Won 11 Oscars. Another 111 wins & 62 nominations.",
  Poster: "http://ia.media-imdb.com/images/M/MV5BMjEzMDNDMDN15BM15BanBnXkFtZTcwMzkyOTUwNw@@._V1_SX300.jpg",
  Metascore: "74",
  imdbRating: "7.7",
  imdbVotes: "606,110",
  imdbID: "tt0120338",
  Type: "movie",
  Response: "True"
}
```

Figure 15 Snapshot of JSON response by web service

In the search parameters we have requested “t=titanic” which is the title of the movie if we change it to “Batman” we will get results for batman. Now our objective is to make this URL request in android framework and get Title, Year and Genre from this data and send it to MovieDetailsView to display it using intents (which we have already done in section 4.2.3).

5.1 Permission

Since we will be using internet from user's device therefore we must ask user's permission to use internet. Add following line in your Application Manifest where parent tag is manifest.

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

5.2 API Call and Response

Following part of the url request will always remain same when searching for title.

URL: <http://www.omdbapi.com/?t=>

Now we need one input field for the user where title is typed and we add that to this url after "t=". Create a UI which has following three items

- EditText box
- Button to search
- TextView to display result.

Sample UI in the figure. Create a controller for this. Let's name it SearchActivity.

Note: In MainActivity add intent to go to search activity when Search Button is clicked also add newly created activity to your manifest file.
Code for Search Activity :

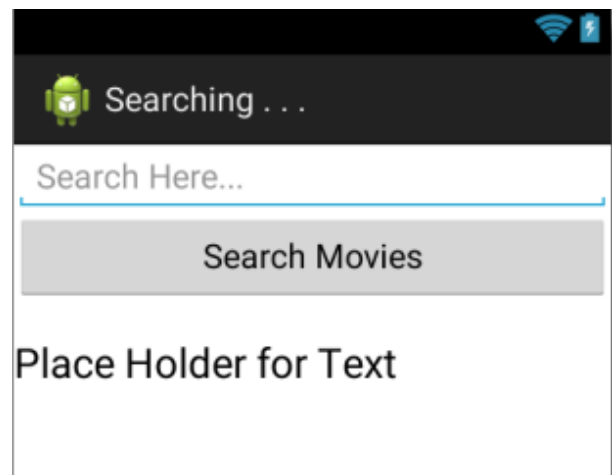


Figure 16 Search Activity Layout

```
EditText et_Search;
Button btn_Search;
TextView tv_Result;

@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_search);
    et_Search = (EditText) findViewById(R.id.et_searchbox);
    btn_Search = (Button) findViewById(R.id.BTN_SearchQueryButton);
    tv_Result = (TextView) findViewById(R.id.tv_SearchResult);
}

@Override
protected void onStart() {
    // TODO Auto-generated method stub
    super.onStart();
    btn_Search.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            if (et_Search.getText().toString().length() > 0) {
                searchTitle(et_Search.getText().toString());
            } else {
                Toast.makeText(SearchActivity.this, "Search Box empty",
                    Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

In `onClick()` method of Search Movies button (`btn_Search`) first we check if the `EditText` is not empty than calling our own method called `searchTitle(String)` which takes a string parameter containing the title user has entered.

In the `searchTitle` method first we will check if the title has some spaces then replace it with `%20` in order to get correct URL format. Then we will create the search URL string and pass the URL to our `SearchTask`, delegate this current activity so that results are sent back to `SearchActivity` and execute the task.

```
protected void searchTitle(String title) {
    title = title.replace(" ", "%20");

    String url = "http://www.omdbapi.com/?t=" + title;
    SearchTask searchTask = new SearchTask();
    searchTask.delegate = this;
    searchTask.execute(url);

    Toast.makeText(getBaseContext(), R.string.searching,
        Toast.LENGTH_LONG)
        .show();
}
```

In order to get the response back to SearchActivity we will create an interface which will be implemented in SearchActivity class. Now create a new java class and name it AsyncResponse. Copy the follow code to this class.

Note: Change package name to yours.

```
package fi.metropolia.qaisersiddique;

public interface AsyncResponse {
    void processFinish(String result);
}
```

After creating the interface we will implement it to SearchActivity by adding " implements AsyncResponse " in class definition.

```
public class SearchActivity extends Activity implements
AsyncResponse{
```

Final step is to write processFinish(String) method in the SearchActivity class.

```
@Override
    public void processFinish(String searchResults) {
        // TODO Auto-generated method stub
        tv_Result.setText(searchResults);
    }
```

In this method we are getting the search results in string format and displaying the raw JSON in the text view of Search Activity for now. Later we will change the code and in processFinish()

method we will send raw JSON to our Parser which in return will give an object of Movie, to be used to send the details to MovieDetailsView.

Finally in order to have Search functionality complete we will write our SearchTask which is AsyncTask. Create a new java file in the project and copy the following code. SearchTask is an AsyncTask which runs in the background and does the network operations so that main UI thread is not blocked (Multi-Threading). If we try to make network calls Android Framework will generate error messages due to strict policy of not doing any network operations in your main thread.

```
package fi.metropolia.kaisersiddique;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URI;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import android.os.AsyncTask;

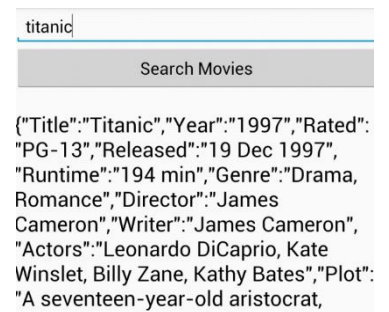
public class SearchTask extends AsyncTask<String, Void, String> {
    public AsyncResponse delegate = null;
    private StringBuffer sb;

    protected String doInBackground(String... params) {
        String url = params[0];
        try {
            HttpClient client = new DefaultHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI(url));
            HttpResponse response = client.execute(request);
            BufferedReader in = new BufferedReader(new
InputStreamReader(
                response.getEntity().getContent()));
            sb = new StringBuffer("");
            String line = "";
            while ((line = in.readLine()) != null) {
                sb.append(line);
            }
            in.close();
            System.out.println(sb.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return sb.toString();
    }
    protected void onPostExecute(String result) {
        delegate.processFinish(result);
    }
}
```

`doInBackground` will be called, when `searchTask` is executed and we send a `String url` as the parameter. Using `HttpClient` and `HttpGet` we fetch the `HttpResponse` object. Then using the `String` builder response is converted to `String` and we return response to `onPostExecute` method in form of a string.

`onPostExecute` is called when background process is finished. In this method we take the result (a string) and send it `processFinished()` delegated to this call via interface.

At this point application should run without errors and display results as shown in screenshot below. Now we will make changes to our `processFinish()` method in `SearchActivity` and parse the JSON to create `Movie` Object.



```
{
  "Title": "Titanic",
  "Year": "1997",
  "Rated": "PG-13",
  "Released": "19 Dec 1997",
  "Runtime": "194 min",
  "Genre": "Drama, Romance",
  "Director": "James Cameron",
  "Writer": "James Cameron",
  "Actors": "Leonardo DiCaprio, Kate Winslet, Billy Zane, Kathy Bates",
  "Plot": "A seventeen-year-old aristocrat,
```

Figure 17 JSON response in TextView

5.3 Parsing JSON and Custom Objects

Let's first change `processFinished` method remove call to `setText` and add calls to 2 new methods.

```
@Override
public void processFinish(String searchResults) {
    Movie movie = createMovieObjectFromJSON(searchResults);
    displayMovieObject(movie);
}

private Movie createMovieObjectFromJSON(String searchResults)
{
    return null;
}

private void displayMovieObject(Movie movie) {
}
```

Our target is to take Raw JSON, Parse it into Movie Object. Now we will create our Movie class with three properties name, year and genre. Create a new java class called Movie and copy this code.

```
package fi.metropolia.qaisersiddique;

public class Movie {

    public String name, year, genre;

    public Movie(String name, String year, String genre) {
        super();
        this.name = name;
        this.year = year;
        this.genre = genre;
    }

    public String getName() {
        return name;
    }

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

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

}
```

Now in the createMovieObjectFromJSON method we will send String and get a movie object in return.

```
private Movie createMovieObjectFromJSON(String searchResults)
{
    String name, year, genre;
    Movie movie = null;
    try {
        JSONObject jsonObject = new
JSONObject(searchResults);
        if (jsonObject.has("Title")) {
            name = jsonObject.get("Title").toString();
        } else {
            name = "Name not found";
        }
        if (jsonObject.has("Year")) {
            year = jsonObject.get("Year").toString();
        } else {
            year = "Year not found";
        }
        if (jsonObject.has("Genre")) {
            genre = jsonObject.get("Genre").toString();
        } else {
            genre = "Genre not found";
        }

        movie = new Movie(name, year, genre);
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return movie;
}
```

The movie object returned from previous method will go to displayMovieObject() method which will take the information and print on the screen for user .

```
private void displayMovieObject(Movie movie) {
    String result;
    result = "Found " + movie.getName()+ "
(+movie.getYear()+)";
    tv_Result.setText(result);
}
```

Note: Use all strings from String.xml instead of hardcoding like in the snippets above. This was used just to make it easier to understand.

Task: Add a View Details Button to Search Activity which will be Invisible until the displayMovieObject() method is called. Once it's called then View Details button should be visible and onClick method should take user to MovieDetailsView Activity and send movie object's properties using intents as done in section 4.2.2.

Hint: You can hide widget in both xml and Java. Keywords are "Visibility GONE".

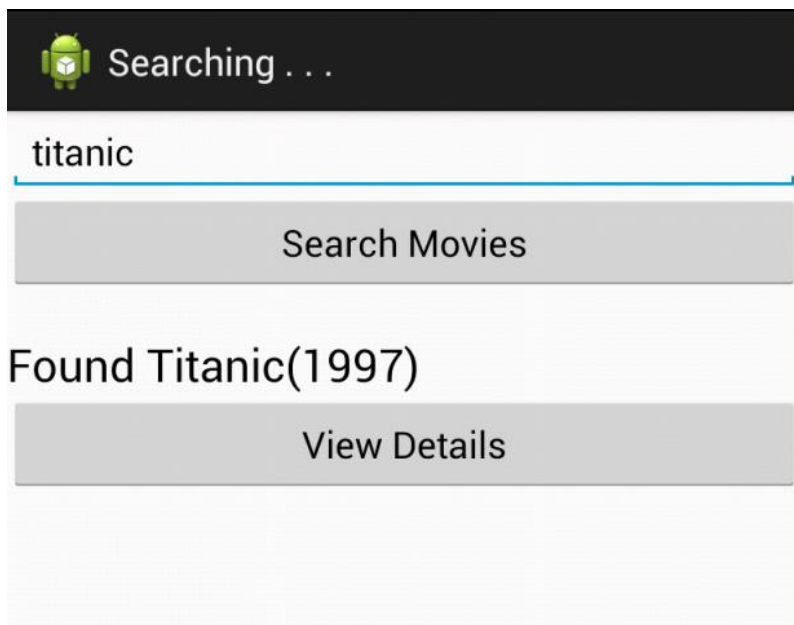


Figure 18 Search result shown

After finishing the TASK View Details button should take you to MovieDetailsView activity where name, year and genre is displayed.
Compile and Run.

TASK: Apart from name year and genre take another value from JSON data of your choice and display it.

6. Data Persistence

At this point, your application has several functionalities for getting the Data, one is with user input and other is coming from the web service. This data will be lost when application is closed. We now want to implement a feature which will allow us to save the data in the android phone and whenever application is launched again saved data is loaded. In android there are 4 different techniques to persist the data. as shown in figure below.

[Shared Preferences](#)

Store private primitive data in key-value pairs.

[Internal Storage](#)

Store private data on the device memory.

[External Storage](#)

Store public data on the shared external storage.

[SQLite Databases](#)

Store structured data in a private database.

[Network Connection](#)

Store data on the web with your own network server.

In this section we will use SQLite Databases to store our three values. However, our data can be stored using other methods as well.

6.1 SQLite

First we will implement a helper class to manage database creation and version management. You create a subclass implementing onCreate and onUpgrade, and this class takes 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.

Create a new java class and name it MovieSQLiteHelper and copy the following code

```
public class MovieSQLiteHelper extends SQLiteOpenHelper {
    String createSQL = "CREATE TABLE movies (name TEXT, year TEXT,
    genre TEXT)";

    public MovieSQLiteHelper(Context context, String name,
        CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        // Execute the SQL sentence for creating the table
        db.execSQL(createSQL);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {
        db.execSQL("DROP TABLE IF EXISTS movies");
        db.execSQL(createSQL);
    }
}
```

Now we need to implement the trigger point where database operations will start. In section 4.2.3 you implemented a Save Movie button in MovieDetailsView activity. Now call a method in it's on click listener

```
saveButton.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        if(isDataValidated()){
            saveMovieToDataBase();
        } else {
            // Do something
        }

    }

});
```

If our data is validated, which means all three fields are not empty then call the method saveMovieToDataBase() else notify user that some data is missing.

isDataValidated() method can look like this.

```
protected boolean dataValidated() {  
    // if all three fields are not empty then return true  
    // else return false  
    return false;  
}
```

6.1.1 SQLite Writing

At this point we will only run `saveMovieToDatabase()` when data is valid and good to go for storage. In this method we will call our helper class to open the database in writing mode and use Content Values to insert the data in the our table called movies.

Copy the following save data method in your `MovieDetailsView` activity class.

```
protected void saveMovieToDataBase() {  
    MovieSQLiteHelper msdbh;  
    SQLiteDatabase db;  
  
    msdbh = new MovieSQLiteHelper(MovieDetailsView.this,  
    "DBMovie", null, 1);  
    // We open the database in writer mode  
    db = msdbh.getWritableDatabase();  
  
    // Create the record using ContentValues  
    ContentValues newRecord = new ContentValues();  
    newRecord.put("name", mName);  
    newRecord.put("year", mYear);  
    newRecord.put("genre", mGenre);  
    // Insert the record in the database  
    db.insert("movies", null, newRecord);  
    db.close();  
}
```

Now when save button is clicked this method will take name , year and genre of the movie and save it to data base , does not matter if data comes from web service (SearchActivity) or user input (AddMovieActivity)

6.1.2 SQLite Reading

Now we want to be able to read from the data base and show the list of all Movies that user has saved in the data base. Following method will open up the database and read all the rows in “movies” table. We will add this method to SavedMoviesActivity that we will create and link it with the button “Saved Movies” on our MainActivity. For now you can use this method in MovieDetailsView and print the result in Log to be sure if all the data is going to database.

We will use the same SQLite helper class to get the readable database and use Cursor to navigate between the rows.

```
private void readMovieFromDataBase() {
    MovieSQLiteHelper msdbh;
    SQLiteDatabase db;

    msdbh = new MovieSQLiteHelper(MovieDetailsView.this,
    "DBMovie", null, 1);
    db = msdbh.getReadableDatabase();
    Cursor c = db.rawQuery("select * from movies", null);

    if (c.moveToFirst()) {
        // List all results
        do {
            String name = c.getString(0);
            String year = c.getString(1);
            String genre = c.getString(2);
            Log.d("Database Testing", name + year +
genre);
        } while (c.moveToNext());
    }
    db.close();
}
```

7. ListViews and custom Listitems

It's time to create the SavedMoviesActivity and its view. Our target here is to show a list of all the movies that are saved in the database using ListView and Custom Adapter. In this section we will learn how to create custom ListView using ArrayList of custom objects (Movie).

Create a new Activity called "SavedMoviesActivity" and it's UI should have a ListView. Create a new xml file in Layout folder and name it "[*saved_movies_activity*](#)"

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listview">
    </ListView>
</LinearLayout>
```

Now we have a widget for List View, next target is populate the items of the list view.

Note: At this point you should have SavedMoviesActivity created and content view is set to this layout.

Now in the saved movies activity we have to first read the data from data base and then display it. Therefore in the onStart method we add following two lines and later look into the code for these methods.

```
readMovieFromDataBase();
showList();
```

readMovieFromDataBase() can be same as what we had in previous section to check in LogCat. Copy the same method in this Activity and we will modify it for our use. In this method when name, year and genre is read from the data base using Cursor (c.getString(0)) after that we are just printing them in the Log Cat. Now we will create Movie Object using this information. We know that constructor of Movie Class asks for 3 parameters, and each row from database should create separate Movie Object. Therefore we modify readMovieFromData() method as shown below

```
String name = c.getString(0);  
String year = c.getString(1);  
String genre = c.getString(2);  
Movie movie = new Movie(name, year, genre);
```

All the movie objects created here will be in the memory, in order to access them easily we will add them to ArrayList of Movie Objects.

In the class level add following field.

```
ArrayList<Movie> moviesList;
```

Finally in the readMovieFromDatabase() method we initialize add newly created object to this list.

```
moviesList = new ArrayList<Movie>();  
Movie movie = new Movie(name, year, genre);  
moviesList.add(movie);
```

7.1 Creating ListView

In the activity layout file we have a ListView widget with id "listview" as shown below

```
android:id="@+id/listview">
```

Create a ListView object with scope of class, in the Activity.

```
ListView mListView;  
ArrayList<Movie> moviesList;
```

Make sure your content View is set and mListView is references to listview widget in onCreate() method.

```
setContentView(R.layout.saved_movies_activity);  
mListView = (ListView)findViewById(R.id.listview);
```

7.1 Array Adapters

All set to start working on ListView. At this point we have data in moviesList (ArrayList of Movie objects) and view is mListview. Now we will hook both data a view in order to accomplish it android framework provides us ArrayAdapter Class. If we have ArrayList<String>, which is array list of strings we can directly use built in adapters as shown in snippet below . But this is not our requirement.

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
    android.R.layout.simple_list_item_1,string_values);  
  
// Assign adapter to List  
mListView.setAdapter(adapter);
```

If you have a list of Strings you can assign pass that to adapter as third parameter (string_values).

Since we will create our custom Custom adapter there for we add following lines to showList() method.

```
private void showList() {  
  
    ArrayAdapter<Movie> adapter=new MoviesArrayAdapter(this,moviesList);  
    mListview.setAdapter(adapter);  
  
}
```

List get populated when we call setAdapter method pass on our adapter to it. In this case name of our adapter is MoviesArrayAdapter which require Context and ArrayList<Movies> as parameters. Now create a new java file in your package and name it MoviesArrayAdapter.

7.2 Custom Array Adapter

We extend this class as shown below.

```
public class MoviesArrayAdapter extends ArrayAdapter<Movie>
```

Now we will write the code in such a way that this adapter can be used for any activity to display list of movies. Therefore, we will require only two parameter to construct it , which is Activity Context and ArrayList of Movie objects. Add the following properties and constructor in the in your class.

```
Activity context;
ArrayList<Movie> moviesList;

public MoviesArrayAdapter(Activity context, ArrayList<Movie>
moviesList) {

    super(context, R.layout.row_layout, moviesList);
    this.context = context;
    this.moviesList = moviesList;
}
```

With this constructor application will know that from which activity you have requested, what layout to use when display a row of list view and an ArrayList that contains all the data. Now we will create a custom layout for each row item and name it row_layout. Create a new xml layout file in Res > Layout folder and add following code to it.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/row_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Movie Name"
        android:textAppearance="?android:attr/textAppearanceLarge"
    />

    <TextView
        android:id="@+id/row_year"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="YEAR"
        android:textAppearance="?android:attr/textAppearanceSmall"
    />

    <TextView
        android:id="@+id/row_genre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Genre"
        android:textAppearance="?android:attr/textAppearanceSmall"
    />

</LinearLayout>
```

This layout will have three text view for each property of the Movie Object and final result will look like the figure below.



Figure 19 Custom list View of saved movies

At this point we have our Adapter ready and now we need to add data to each row and return it. For which we will use getView method of ArrayAdapter Class to inflate our row_layout and use TextViews defined in it.

```
public View getView(int position, View convertView, ViewGroup
parent) {

    LayoutInflater inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

    View rowView = inflater.inflate(R.layout.row_layout, parent, false);

    return rowView;

}
```

First LayoutInflater is created and then used to inflate R.layout.row_layout. In doing so, we will get access to our TextView defined in row_layout and setText to it.

After rowView is initialized add following lines.

```
TextView nameView = (TextView) rowView.findViewById(R.id.row_name);
    TextView yearView = (TextView)
rowView.findViewById(R.id.row_year);
    TextView genreView = (TextView)
rowView.findViewById(R.id.row_genre);

nameView.setText(moviesList.get(position).getName());
yearView.setText(moviesList.get(position).getYear());
genreView.setText(moviesList.get(position).getGenre());
```

This completes our custom ArrayAdapter and all the data from database should be visible in SavedMoviesActivity.

7.3 List item Click Listener

Now we have our listview created and populated and we want to create functionality to do some thing when item is clicked for example delete an item update an item or go to MovieDetailsView.

In the onStart Method of SavedMoviesActivity add onItemClickListener as shown below.

```
mListView.setOnItemClickListener(new OnItemClickListener() {

@Override
public void onItemClick(AdapterView<?> parent, View view, int
position, long id)
{

Toast.makeText(getApplicationContext(),
moviesList.get(position).getName(), Toast.LENGTH_SHORT).show();

}
});
```

Index of the row in the mListView and index of item in moviesList is our link between the view and model. Therefore an item at index 3 will be same on list view and array list. In onItemClickListener we get the position integer and use that to get the movie object, which can be used to access all the properties of that Movie object. In the code above we are calling getName() method of Movie.

TASK: Create a functionality to `onItemClickListener`, anything you like. It could be taking user to `MovieDetailsView` and adding a delete functionality there instead of `Save`, since item is already saved (Hint: You can use a key value pair in `putExtra` which can be used to enable delete button) or you can create a dialog alert showing the details of the movie with option to delete the movie.

8. Dialogue Alert

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.. If you decide to implement delete functionality using dialogue box, following snippet can be used to create your method.

```
protected void showDialogue(Movie movie) {
    // TODO Auto-generated method stub
    AlertDialog.Builder builder = new
AlertDialog.Builder(this);
    // Add the buttons
    builder.setPositiveButton(android.R.string.ok, new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
int id) {
            // User clicked OK button
        }
    });
    builder.setNegativeButton("Delete", new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
int id) {
            // User clicked Delete button
        }
    });
    builder.setMessage(movie.getName() + " " +
movie.getYear()+ movie.getGenre());
    AlertDialog dialog = builder.create();
    dialog.show();
}
```


9. Google Maps

Creating a new Android application that uses the Google Maps Android API v2 requires several steps. Many of the steps outlined in this section will only have to be performed once, but some of the information will be a handy reference for future applications. The overall process of adding a map to an Android application is as follows:

- Download and configure the Google Play services SDK, which includes the Google Maps Android API. If you use the Google Maps Mobile SDK for Work you must download and configure the Google Maps Mobile SDK for Work static library.
- Obtain an API key. To do this, you will need to register a project in the Google APIs Console, and get a signing certificate for your app.
- Add the required settings in your application's manifest.
- Add a map to your application.
- Publish your application.

9.1 Setting up Google Play Services

To make the Google Play services APIs available to your app:

- Copy the library project at `<android-sdk>/extras/google/google_play_services/libproject/google-play-services_lib/` to the location where you maintain your Android app projects.
- Import the library project into your Eclipse workspace. Click **File > Import**, select **Android > Existing Android Code into Workspace**, and browse to the copy of the library project to import it.
- In your app project, reference Google Play services library project. To add a reference to a library project, follow these steps:
 - Make sure that both the project library and the application project that depends on it are in your workspace. If one of the projects is missing, import it into your workspace.
 - In the **Package Explorer**, right-click the dependent project and select **Properties**.
 - In the **Properties** window, select the "Android" properties group at left and locate the **Library** properties at right.
 - Click **Add** to open the **Project Selection** dialog.
 - From the list of available library projects, select a project and click **OK**.
 - When the dialog closes, click **Apply** in the **Properties** window.
 - Click **OK** to close the **Properties** window.

Note: You should be referencing a copy of the library that you copied to your development workspace—you should not reference the library directly from the Android SDK directory.

- After you've added the Google Play services library as a dependency for your app project, open your app's manifest file and add the following tag as a child of the `<application>` element:

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

9.2 Displaying the debug certificate fingerprint

Firstly we need the *fingerprint* of our debug keystore, which is needed for getting the key for using Google Maps. To do this we have to execute the following command in a console:

```
keytool -list -v -keystore
"C:\Users\your_user_name\.android\debug.keystore" -alias
androiddebugkey -storepass android -keypass android
```

Please note that the user directory has to be changed in the command. Keytool is a command included in Java JDK, so the “*path*” environment variable has to be setup correctly.

We have to copy the string in format SHA1, which is something like this: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75. It is important to highlight that each computer will have different numbers. The SHA1 can also be obtained from Eclipse in **Window -> Preferences -> Android -> Build**.

9.3 Obtaining an API Key

Obtain Key from here :

```
https://code.google.com/apis/console/?noredirect
```

If your application is registered with the Google Maps Android API v2 service, then you can request an API key. It's possible to register more than one key per project.

- Navigate to your project in the Google APIs Console.
- In the **Services** page, verify that the "Google Maps Android API v2" is enabled.
- In the left navigation bar, click **API Access**.
- In the resulting page, click **Create New Android Key....**
- In the resulting dialog, enter the SHA-1 fingerprint, then a semicolon, then your application's package name. For example:

```
BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;com.example.android.mapexample
```

- The Google APIs Console responds by displaying **Key for Android apps (with certificates)** followed by a forty-character API key, for example:

```
AIzaSyBdV1-cTICSwYKrZ95SuvNw7dbMuDt1KG0
```

9.4 Add API Key to your application

Follow the steps below to include the API key in your application's manifest, contained in the file **AndroidManifest.xml**. From there, the Maps API reads the key value and passes it to the Google Maps server, which then confirms that you have access to Google Maps data.

In **AndroidManifest.xml**, add the following element as a child of the `<application>` element, by inserting it just before the closing tag `</application>`:

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY" />
```

Substitute your API key for **API_KEY** in the value attribute. This element sets the key `com.google.android.maps.v2.API_KEY` to the value of your API key, and makes the API key visible to any `MapFragment` in your application.

Save **AndroidManifest.xml** and re-build your application.

9.5 Setting up Application and Manifest

An Android application that uses the Google Maps Android API should specify the following settings in its manifest file, **AndroidManifest.xml**:

- A reference to the Google Play services version. If you have followed the steps on this page up to this point, you have already added the required declaration to your application manifest.
- The Maps API key for the application. The key confirms that you've registered with the Google Maps service via the Google APIs Console. If you have followed the steps on this page up to this point, you have already added the API key to your application manifest.
- Permissions that give the application access to Android system features and to the Google Maps servers. See below for instructions on adding this setting.
- (Recommended) Notification that the application requires OpenGL ES version 2. External services can detect this notification and act accordingly. For example, Google Play Store won't display the application on devices that don't have OpenGL ES version 2. See below for instructions on adding this setting.

Now we will add permissions and notification that application requires OpenGL ES Version 2 in AndroidManifest.xml as a child of manifest tag file.

```
<permission  
  android:name="com.example.mapdemo.permission.MAPS_RECEIVE"  
  android:protectionLevel="signature"/>  
<uses-permission  
  android:name="com.example.mapdemo.permission.MAPS_RECEIVE" />
```

And we also need to add these other permissions:

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission  
  android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission  
  android:name="com.google.android.providers.gsf.permission.READ_GSERVICES  
"/>  
<uses-permission  
  android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission  
  android:name="android.permission.ACCESS_FINE_LOCATION" />  
<uses-permission  
  android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Finally we add

```
<uses-feature  
android:glEsVersion="0x00020000"  
android:required="true"/>
```

9.6 Adding the Map to your Application

The easiest way to test that your application is configured correctly is to add a simple map. You will have to make changes in two files: the XML file that defines the app's layout, and the main activity Java file.

Add the following fragment in the app's layout XML file. If you created a 'hello world' app using the Android Developer Tools (ADT) package in Eclipse, the file is at res/layout/activity-main.xml. Replace the entire contents of that file with the following code.

```
<?xml version="1.0" encoding="utf-8"?>  
<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/map"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:name="com.google.android.gms.maps.SupportMapFragment"/>
```

Compile and run the code.

9.7 Using Google Map Class

Once the map is working you can try following code in your onCreate method of MainActivity to set the Map View to specific coordinates, add markers, zoom and etc.

```
double latitude = 60.221058;  
double longitude = 24.805060;  
  
String name = "Intro to Android Course Location";  
GoogleMap map = ((SupportMapFragment)  
getSupportFragmentManager()  
    .findFragmentById(R.id.map)).getMap();  
map.addMarker(new MarkerOptions().position(  
    new LatLng(latitude,  
longitude)).title(name));  
LatLng latLng = new LatLng(latitude, longitude);  
map.moveCamera(CameraUpdateFactory.newLatLng(latLng));  
map.animateCamera(CameraUpdateFactory.zoomTo(15), 2000,  
null);
```

10. Geocoding

We will send a URL to SearchTask which will return us JSON and parse that JSON differently, in order to get the co ordinates for the typed address.

API Call : <http://maps.googleapis.com/maps/api/geocode/json?address=>

JSON Parser can look like this

```
if (status.equals("OK"))
{
JSONArray array = jsonObject.getJSONArray("results");
JSONObject item = array.getJSONObject(0);

JSONObject point =
item.getJSONObject("geometry").getJSONObject("location");

coordinates[0] = point.getDouble("lat");
coordinates[1] = point.getDouble("lng");

System.out.println("Latitude: " + coordinates[0] + " - Longitude: " +
coordinates[1]);
}
```

11. Location Services (Optional)

For accessing the location systems we need to use *LocationManager*, which is obtained with *LOCATION_SERVICE*.

Firstly we have to create two attributes as follows:

```
LocationManager mylocManager;  
LocationListener mylocListener;
```

LocationManager will allow to access the location service and *LocationListener* will be the class that will manage the different location events.

Then we have to write the following line of code in the “*onResume*” method.

```
mylocManager = (LocationManager)  
getSystemService(Context.LOCATION_SERVICE);
```

Now we instantiate the *MyLocationListener* class, which will implement *LocationListener*, which will handle the different location events.

```
mylocListener = new MyLocationListener();
```

The following line of code will register the listener receiving the coordinates from the device.

```
mylocManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,  
0, 0, mylocListener);
```

The last line of code receives coordinates by using the network of the phone (GPRS, 3G, etc.). If we want to use the GPS, we have to use the following code:

```
mylocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  
0, 0,  
mylocListener);
```

We could change between systems using the different parameters:

LocationManager.GPS_PROVIDER and
LocationManager.NETWORK_PROVIDER.

The “*onResume*” method should be as follows:

```
@Override
public void onResume()
{
    super.onResume();
    mylocManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);
    mylocListener = new MyLocationListener();
    mylocManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, mylocListener);
}
```

On the other hand, in the “*onPause*” method we have to remove the listener because if the application is closed or the activity is in paused state, the listener continues working. So if we do not do this, the battery of the device will run out quickly.

```
@Override
public void onPause()
{
    super.onPause();
    mylocManager.removeUpdates(mylocListener);
}
```

Finally, we have to create the class that implements the *LocationListener*. It will have the methods for receiving the coordinates.


```
public class MyLocationListener implements LocationListener {
    @Override
    public void onLocationChanged(Location loc) {
        String coordinates = "My coordinates are: " + "Latitude = "
        + loc.getLatitude() + "- Longitude = " + loc.getLongitude();
        Toast.makeText(getApplicationContext(), coordinates,
        Toast.LENGTH_LONG).show();
    }
    @Override
    public void onProviderDisabled(String provider) {
        Toast.makeText( getApplicationContext(), "Gps
        Disabled", Toast.LENGTH_SHORT ).show();
    }
    @Override
    public void onProviderEnabled(String arg0) {
        Toast.makeText( getApplicationContext(), "Gps
        Enabled", Toast.LENGTH_SHORT ).show();
    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle
    extras) {
        // TODO Auto-generated method stub
    }
}
```

The “*onLocationChanged*” method allows receiving the location coordinates. It is important to highlight that the system uses the coordinates system called “geodesic decimal degrees”, but many other information systems use other different location systems.

The “*getLatitude()*” and “*getLongitude()*” methods return the latitude and longitude, respectively. The first one is the distance from the Equator until the position of the coordinate. If the place is in the north this value will be positive, if instead it is in the south it will be a negative value. The longitude is the distance from the Greenwich meridian: if the place is to the left, it will be a negative value; and positive if the position is to the right of the meridian.

```
loc.getLatitude();
loc.getLongitude();
```

The “*onProviderEnabled*” and “*onProviderDisabled*” methods detect if the location system is enabled or disabled.

The “*onStatusChanged*” method allows knowing the state of the GPS, according to the following values:

```
public static final int OUT_OF_SERVICE = 0;  
public static final int TEMPORARILY_UNAVAILABLE = 1;  
public static final int AVAILABLE = 2;
```

Finally, we have to add this line to the manifest file. It is the permission for accessing the location systems of the device. We will add this line in the “*AndroidManifest.xml*” file, before the “*application*” node.

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

12. Menus

When we created the project, Android SDK created required methods for us in order to program options menu. These methods were located in MainActivity. Our task here is to take exit button to options menu. Therefore let's look how options menu can be programmed.

onCreateOptionsMenu(Menu menu) inflates the menu xml file which is located in Res > menu > main.xml

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
```

The second method is where we define what happens when an item from option menu is clicked.

```
@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar
        will
        // automatically handle clicks on the Home/Up button, so
        long
        // as you specify a parent activity in
        AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

Finally we will edit the main.xml file to add, edit or remove menu items. Here you can add more items and use their id as a reference to find out when that particular item is clicked.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="fi.metropolia.qaisersiddique.MainActivity" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />

</menu>
```

13. Supporting Multiple Screen Sizes (Android.com, 2014)

Android categorizes device screens using two general properties: size and density. You should expect that your app will be installed on devices with screens that range in both size and density. As such, you should include some alternative resources that optimize your app's appearance for different screen sizes and densities.

- There are four generalized sizes: small, normal, large, xlarge
- And four generalized densities: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)

To declare different layouts and bitmaps you'd like to use for different screens, you must place these alternative resources in separate directories, similar to how you do for different language strings.

Also be aware that the screens orientation (landscape or portrait) is considered a variation of screen size, so many apps should revise the layout to optimize the user experience in each orientation.

13.1 Create Different Layouts

To optimize your user experience on different screen sizes, you should create a unique layout XML file for each screen size you want to support. Each layout should be saved into the appropriate resources directory, named with a `-<screen_size>` suffix. For example, a unique layout for large screens should be saved under `res/layout-large/`.

Note: Android automatically scales your layout in order to properly fit the screen. Thus, your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout structure that affects the user experience (such as the size or position of important views relative to sibling views).

For example, this project includes a default layout and an alternative layout for *large* screens:

```
MyProject/  
  res/  
    layout/  
      main.xml  
    layout-large/  
      main.xml
```

The file names must be exactly the same, but their contents are different in order to provide an optimized UI for the corresponding screen size.

Simply reference the layout file in your app as usual:

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

The system loads the layout file from the appropriate layout directory based on screen size of the device on which your app is running. More information about how Android selects the appropriate resource is available in the [Providing Resources](#) guide.

As another example, here's a project with an alternative layout for landscape orientation:

```
MyProject/  
  res/  
    layout/  
      main.xml  
    layout-land/  
      main.xml
```

By default, the layout/main.xml file is used for portrait orientation.

If you want to provide a special layout for landscape, including while on large screens, then you need to use both the large and land qualifier:

```
MyProject/  
  res/  
    layout/           # default (portrait)  
      main.xml  
    layout-land/      # landscape  
      main.xml  
    layout-large/     # large (portrait)  
      main.xml  
    layout-large-land/ # large landscape  
      main.xml
```

Note: Android 3.2 and above supports an advanced method of defining screen sizes that allows you to specify resources for screen sizes based on the minimum width and height in terms of density-independent pixels. This lesson does not cover this new technique. For more information, read [Designing for Multiple Screens](#).

13.2 Create Different Bitmaps

You should always provide bitmap resources that are properly scaled to each of the generalized density buckets: low, medium, high and extra-high density. This helps you achieve good graphical quality and performance on all screen densities.

To generate these images, you should start with your raw resource in vector format and generate the images for each density using the following size scale:

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0 (baseline)
- ldpi: 0.75

This means that if you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices.

Then, place the files in the appropriate drawable resource directory:

```
MyProject/  
  res/  
    drawable-xhdpi/  
      awesomeimage.png  
    drawable-hdpi/  
      awesomeimage.png  
    drawable-mdpi/  
      awesomeimage.png  
    drawable-ldpi/  
      awesomeimage.png
```

Any time you reference **@drawable/awesomeimage**, the system selects the appropriate bitmap based on the screen's density.

Note: Low-density (ldpi) resources aren't always necessary. When you provide hdpi assets, the system scales them down by one half to properly fit ldpi screens.

For more tips and guidelines about creating icon assets for your app, see the [Iconography design guide](#).

14. Important Links

List of some important links which can be followed to learn more about Android application development. Since this was a hands-on course and application was designed for learning basics only, therefore not all the matters were considered. All students should read the following material to familiarize themselves on some aspects that were not considered in this course.

14.1 Android Training

For beginner's Android.com provide tutorials to get started. Go through with Getting Started on following link

<http://developer.android.com/training/index.html>

14.2 User Interfaces

Android.com provide very good material on different UI design patterns

<http://developer.android.com/guide/topics/ui/index.html>

14.3 Best practices

Following best practices should be considered before publishing an app.

Better user experience:

<http://developer.android.com/training/best-ux.html>

Better user interface

<http://developer.android.com/training/best-ui.html>

Best practices for user input

<http://developer.android.com/training/best-user-input.html>

Best practices for background processes

<http://developer.android.com/training/best-background.html>

14.4 Download default icons

Default icons can be downloaded from

<https://developer.android.com/design/downloads/index.html>

14.5 Device art generator

You can generate screen shots of your app into device frames using this web tool

<http://developer.android.com/distribute/tools/promote/device-art.html>

14.6 Creating icons for different screens (Assets studio)

Assets studio can be used to create icons for different screen sizes

<http://romannurik.github.io/AndroidAssetStudio/>

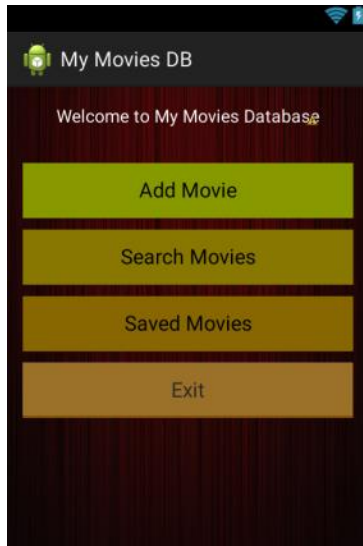
14.7 Publishing check list

Very nice checklist compiled in the following list

<http://developer.android.com/distribute/tools/launch-checklist.html>

Lab 1 Extras Better UI :

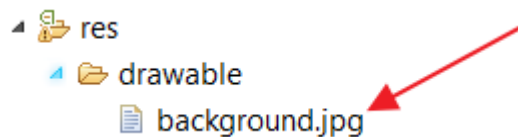
Try to improve the UI as shown in Figure Below



Changing the background of your Main Activity by adding the RelativeLayout element of it's layout xml file.

```
android:background="@drawable/background"
```

This line of code will try to put the background image which will be located in folder res > drawable and name of the file is background.



To Add the image file in your project, go to your workspace folder and locate your project and its res folder, copy and paste your background image there.

To change colour of the Button use following line in your button widget, Colour code can be found here : <http://html-color-codes.info/>

```
android:background="#889900"
```

References

Android.com. (2014, October 18). *Android Developers*. Retrieved from Developer.Android.com:
<http://developer.android.com/training/index.html>

Antonio García Cabot and Eva García López (2014, August 1). Introduction to Android
Programming Course material.

TABLE OF TASKS	2
TASK: READ ACTIVITY LIFE CYCLE (CLICK) AND JUST LIKE ONCREATE AND ONSTART METHODS, CREATE OTHER ACTIVITY LIFECYCLE METHOD E.G ONPAUSE, ONRESUME ADD FOLLOWING LINE OF THE CODE IN ALL OF THE METHODS AND SEE WHAT HAPPENS WHEN YOU RUN THE APP	13
TASK: VALIDATE THE DATA BEFORE SENDING TO THE NEXT ACTIVITY. MAKE SURE NONE OF THE INPUT FIELDS ARE EMPTY.	19
TASK:.....	20
CLEAR BUTTON: WHEN CLEAR BUTTON IS PRESSED, APPLICATION SHOULD KILL CURRENT ACTIVITY AND GO TO MAIN ACTIVITY	20
TASK: ADD A VIEW DETAILS BUTTON TO SEARCH ACTIVITY WHICH WILL BE INVISIBLE UNTIL THE DISPLAYMOVIEOBJECT() METHOD IS CALLED. ONCE IT'S CALLED THEN VIEW DETAILS BUTTON SHOULD BE VISIBLE AND ONCLICK METHOD SHOULD TAKE USER TO MOVIEDetailsVIEW ACTIVITY AND SEND MOVIE OBJECT'S PROPERTIES USING INTENTS AS DONE IN SECTION 4.2.2.	29
TASK: APART FROM NAME YEAR AND GENRE TAKE ANOTHER VALUE FROM JSON DATA OF YOUR CHOICE AND DISPLAY IT.	29
TASK: CREATE A FUNCTIONALITY TO ONITEMCLICKLISTENER, ANYTHING YOU LIKE. IT COULD BE TAKING USER TO MOVIEDetailsVIEW AND ADDING A DELETE FUNCTIONALITY THERE INSTEAD OF SAVE, SINCE ITEM IS ALREADY SAVED (HINT: YOU CAN USE A KEY VALUE PAIR IN PUTEXTRA WHICH CAN BE USED TO ENABLE DELETE BUTTON) OR YOU CAN CREATE A DIALOG ALERT SHOWING THE DETAILS OF THE MOVIE WITH OPTION TO DELETE THE MOVIE.	40