

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. but a link to details is given. All section heading are linked with relative resources on android.com. To get full understanding of the topic developer must read the reference documentation on android.com.

Contents

Preface	1
1. Setting up the Environment	4
1.1 Installation	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 Data Sharing.....	Error! Bookmark not defined.
4.2.1 Creating Input Fields	16
4.2.2 Getting and Sending User Data	17
4.2.3 Retrieve data from Intent and displaying	19
5. Web Services.....	21
5.1 Permission	22
5.2 API Call and Response	22
5.3 Parsing JSON and Custom Objects.....	27
6. Data Persistence	31
6.1 SQLite	31

6.1.1	SQLite Writing	33
6.1.2	SQLite Reading	34
7.	ListViews and custom Listitems	34
7.1	Creating ListView	36
7.1	Array Adapters	36
7.2	Custom Array Adapter	37
7.3	List item Click Listener.....	40
8.	Dialogue Alert	41
9	Menus.....	42
Table of Figures		Error! Bookmark not defined.

1. Setting up the Environment

1.1 Installation

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

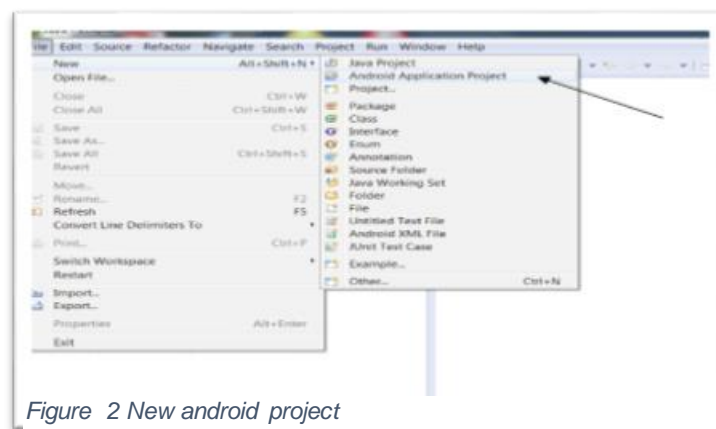


Figure 2 New android project

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.

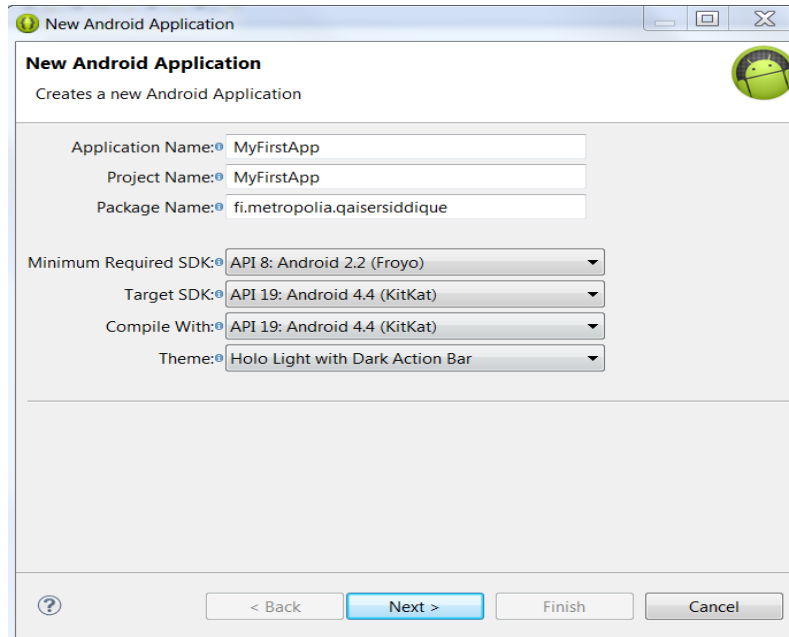


Figure 3 Naming the project

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.

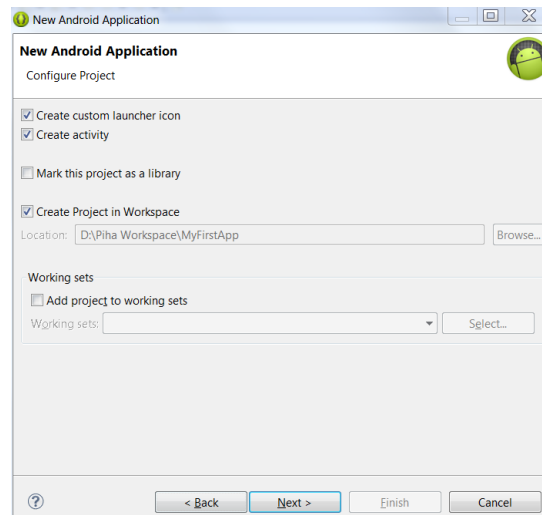


Figure 4 Project configurations

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.

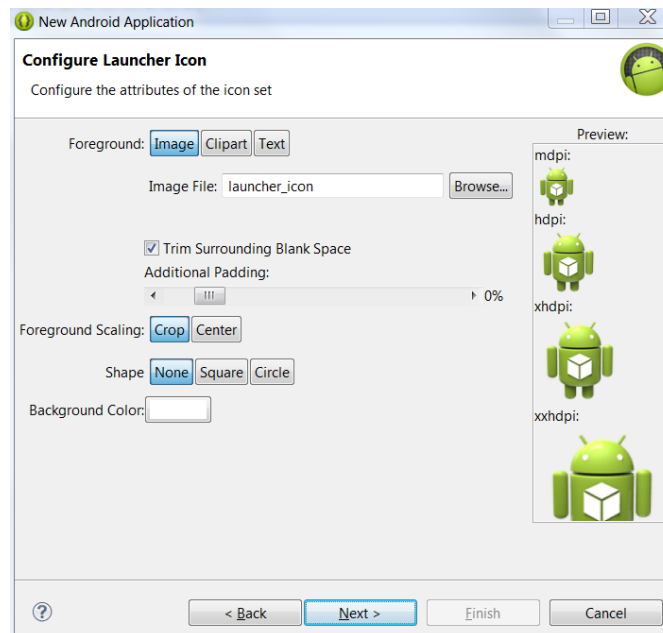


Figure 5 Application icons

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.

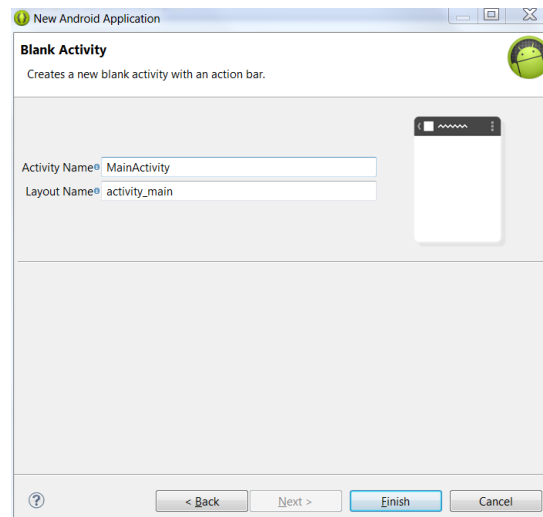


Figure 6 Default activity

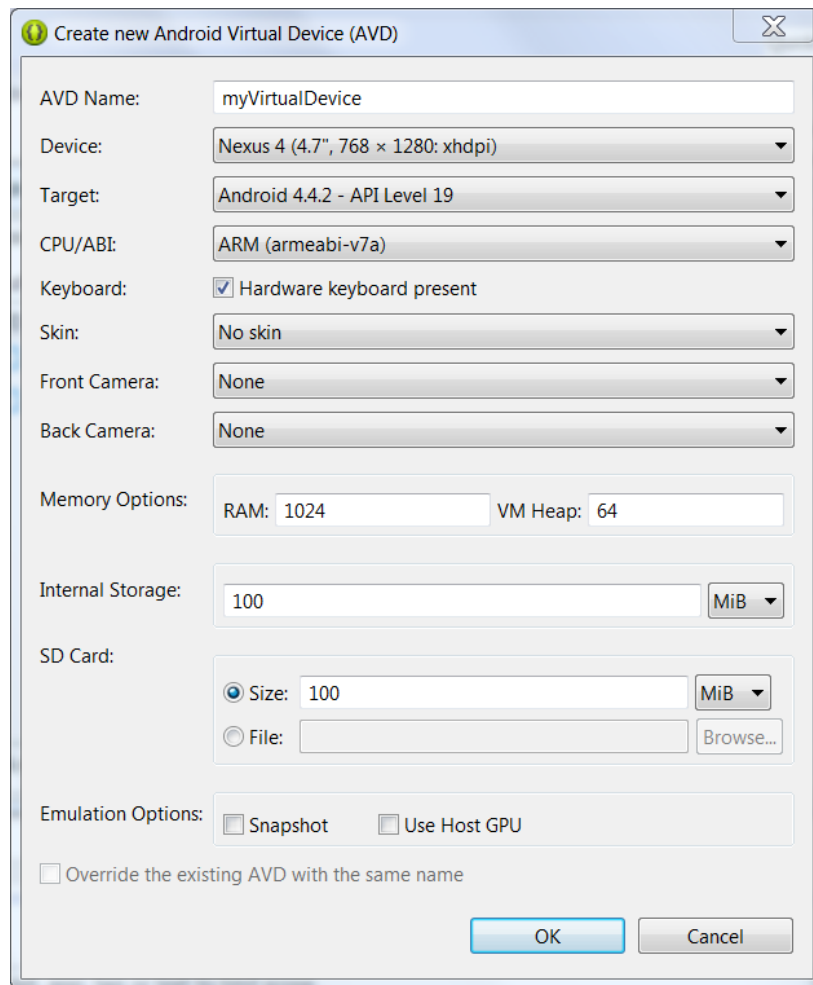
2.2 Creating and running in Emulator

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

Figure 7 Creating Emulator



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.

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.

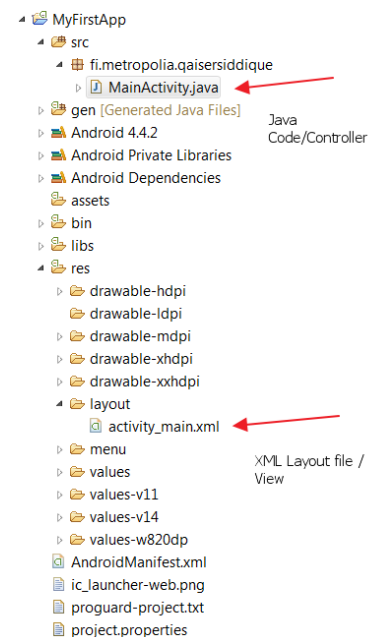


Figure 8 Location Java and XML files

```
<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.

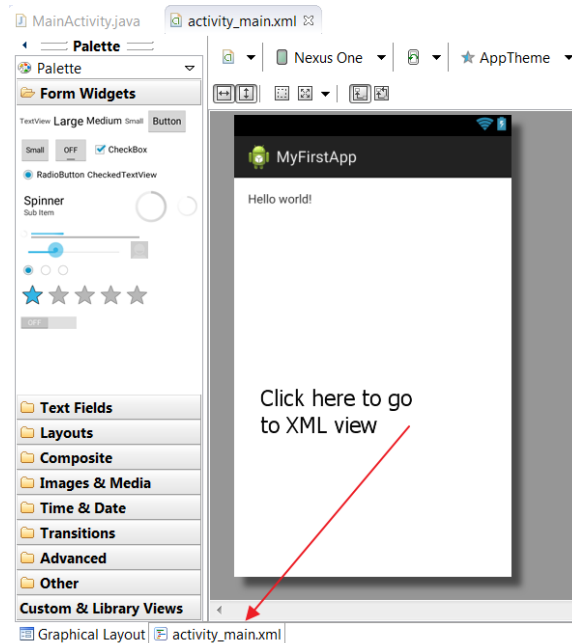


Figure 9 Layout editor

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.

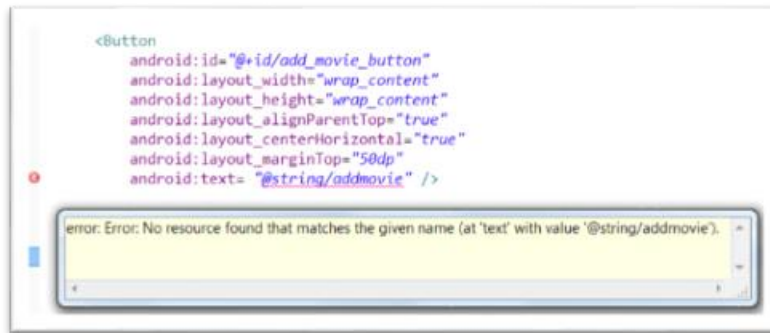


Figure 10 Errors on XML editor

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.

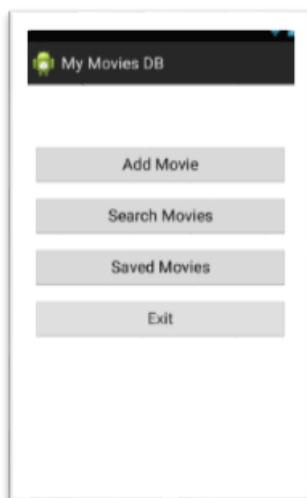


Figure 11 Main Activity layout

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.

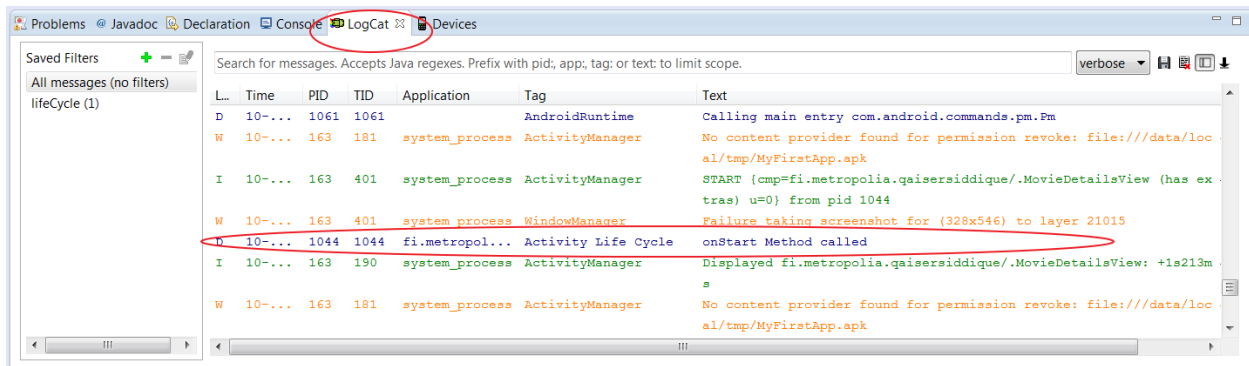


Figure 12 LogCat view

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

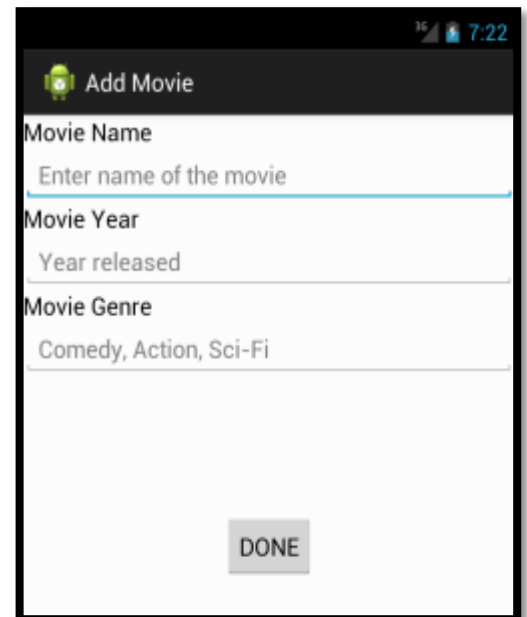


Figure 13 Add Movies Layout


```
<TextView
    android:id="@+id/TV_movieYear"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/movie_name_ET"
    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:layout_alignParentLeft="true"
    android:layout_below="@+id/labelForMovieYear"
    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 displaying

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.

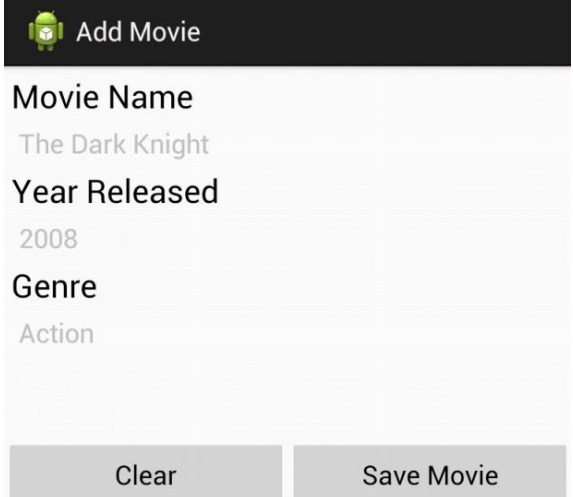


Figure 14 Add Movie layout

```
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.