UNIVERSITY OF NORTHUMBRIA

School of Computing, Engineering and Information Sciences

Applied Computing BSc (Hons)

Name: Bikesh Kawan

Student Number: 14037560

Module Title: Mobile Application Development

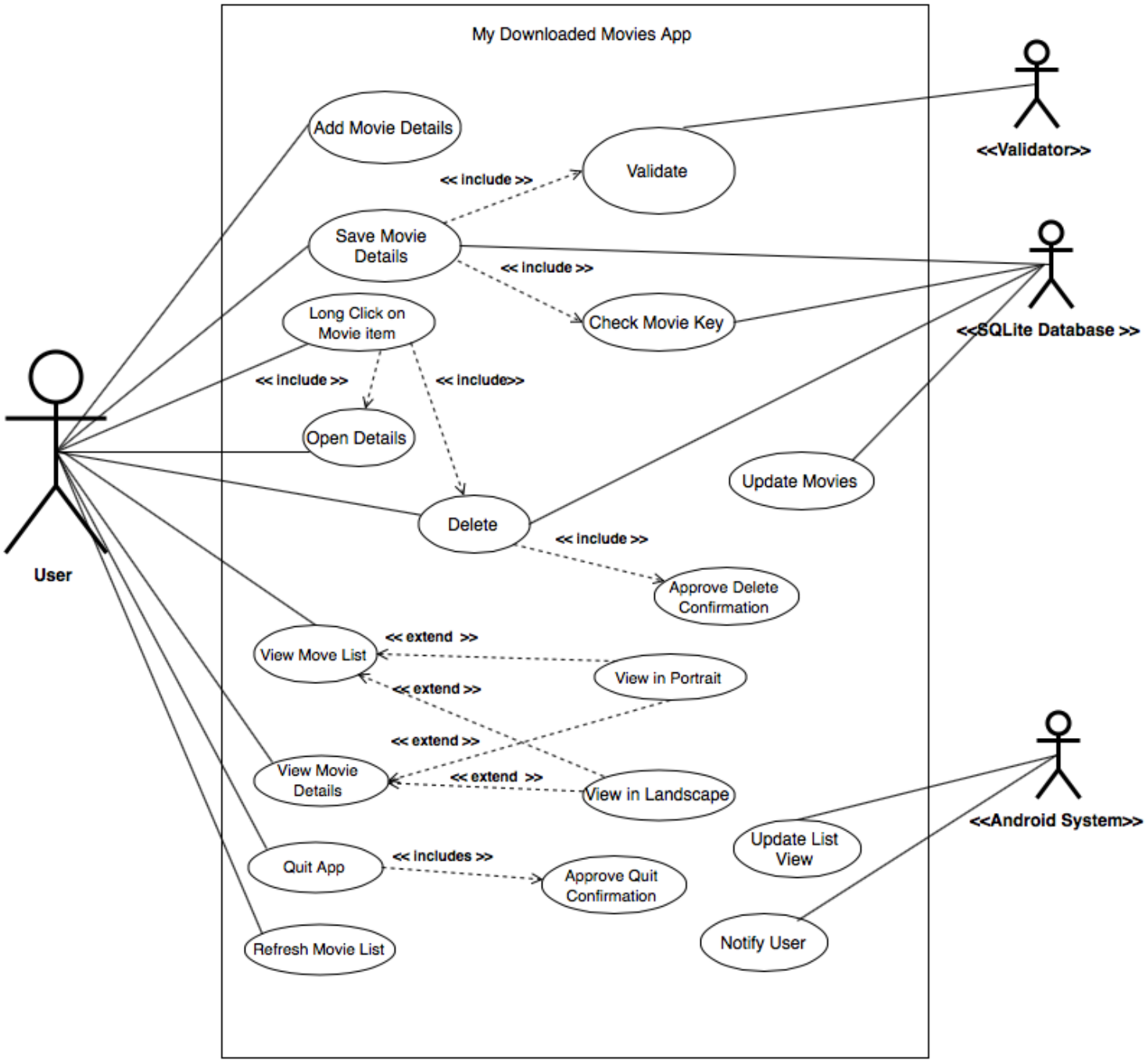
Module Code: CM0573

Submission Date: 08-06-2015

Module Tutor : Li Zhang

Bikesh Kawan, w14037560

1. Use Case Diagrams



1.1 List of Actors involved in use case diagram

1. User
2. Validator
3. SQLite Database
4. Android System

1.2 Brief description of use cases

Add Movie Details:
This use case describes how user adds movie details such as title, language, rating, running time etc.

Save Movie Details:
This use case describes how user saves movie details into database. The use case finishes when movie key is unique and all the input fields are valid.

Long Click on Movie Item:
This use starts when user clicks movie item from the list for a longer time. This use case describes how user selects available options from a dialog box.

Open Details:
This use case describes how user views movie details in landscape or portrait mode.

Delete
This use case describes how user deletes selected movie item from list. The use case finishes when user approve delete confirmation. Then, database deletes the selected movie item from the list.

View Movie List:
This use case describes how user views movie lists in landscape or portrait mode.

View Movie Details:
This use case describes how user views movie details in landscape or portrait mode

View in Portrait:
This use case describes how movie list or movie details displays in portrait mode.

View in Landscape
This use case describes how movie list and movie details displays in landscape mode. Movie list is displayed on right-hand side and movie details are be displayed on left- hand side.

Quit App
This use describes how user quits application. The use case finishes when user approve quit confirmation.

Update list View
This use case describes how android system updates the list view for movie list. Android System updates list view after adding movie successfully into database or delete movie successfully from database.

Notify User
This use case describes how android system notifies user. The android system displays confirmation message after saving movie details or deleting movie.

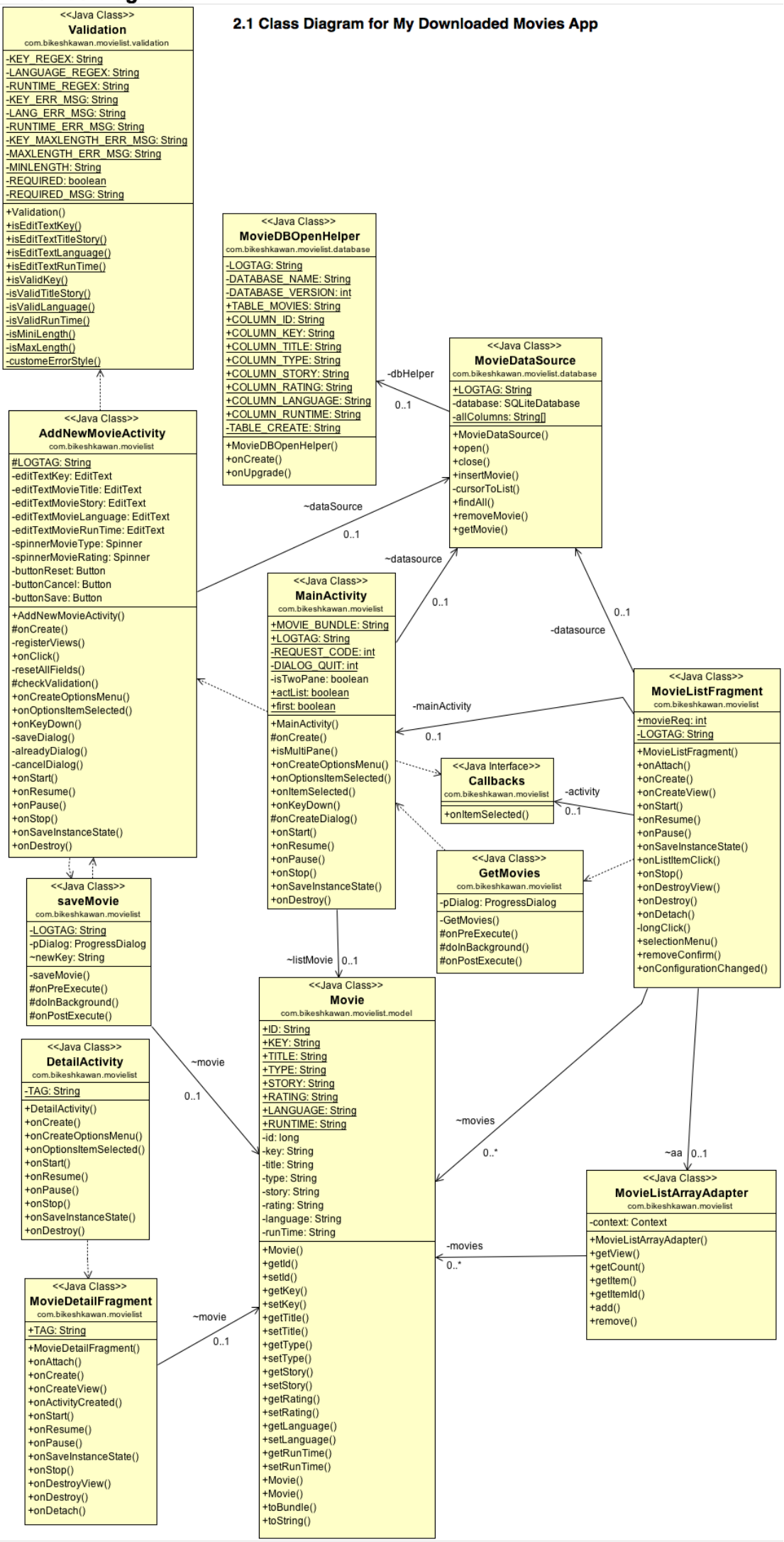
Validate
This use describes how validator checks all the input text before saving movie into database. The validate will display message if there are any errors in input texts.

Check Movie Key:
This use case describes how SQLite Database checks the movie key before saving move details into database. The use case finishes when the movie key is unique.

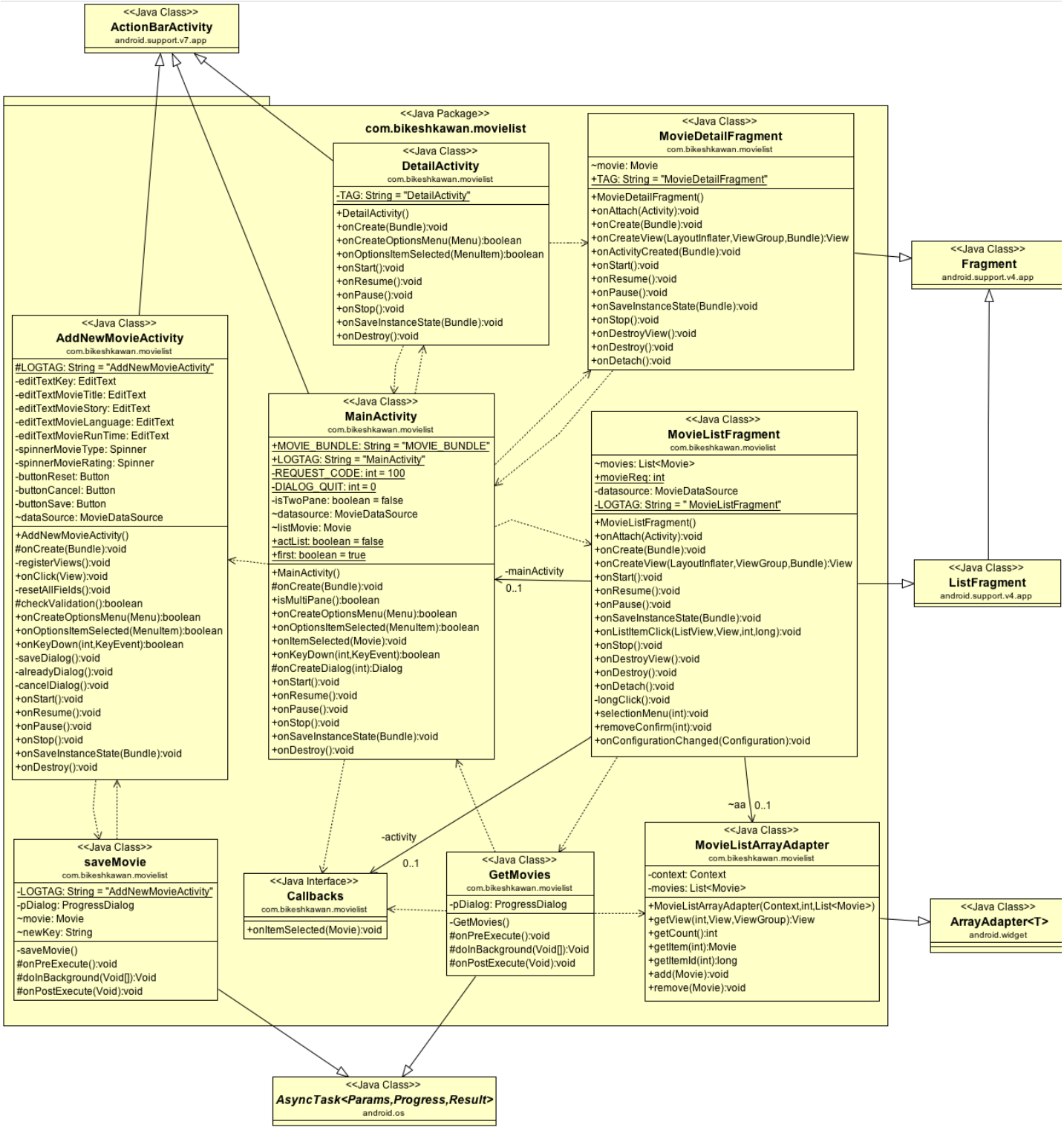
Update Movie
This use case describes how SQLite System updates movies in database. SQLite updates movies when user adds or deletes movies successfully.

2. Class Diagrams

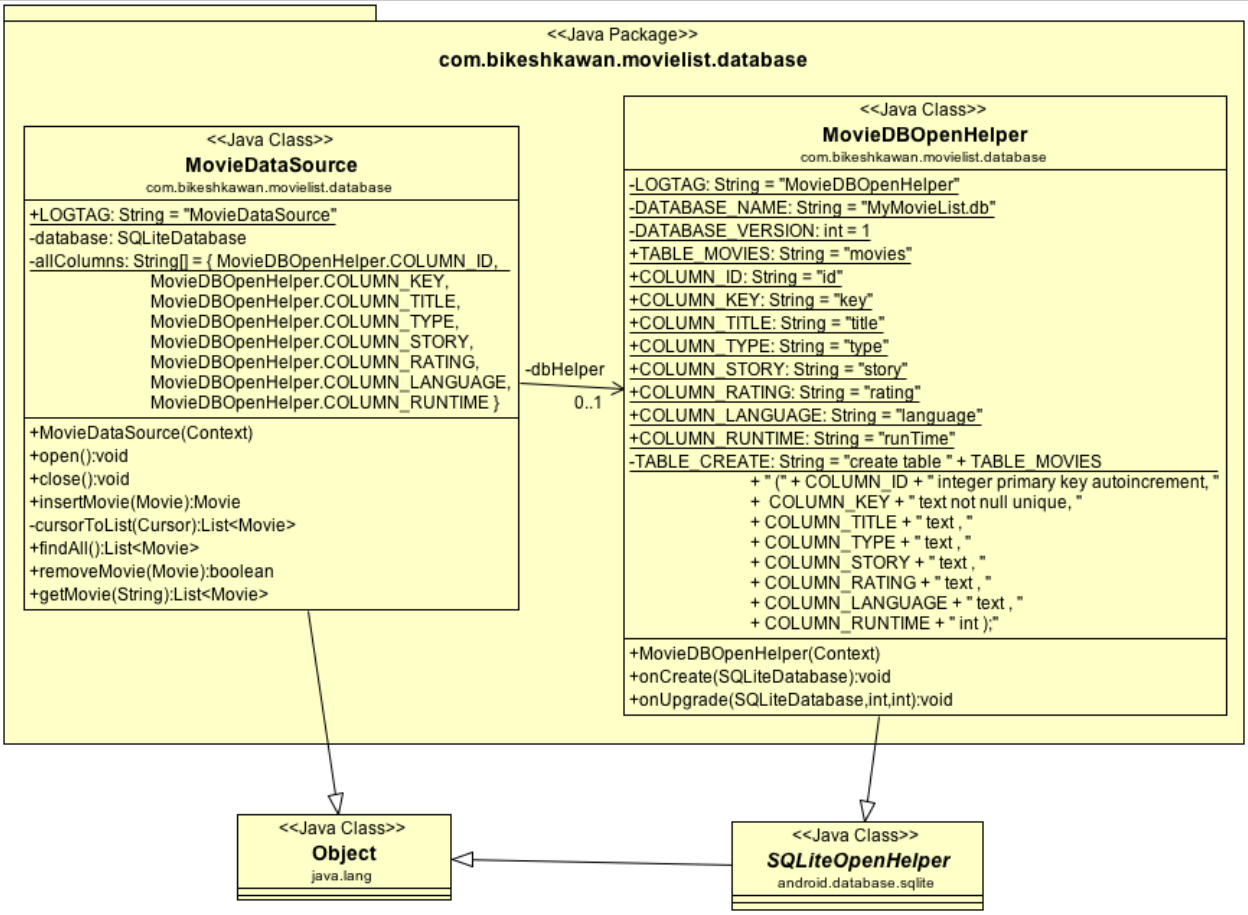
2.1 Class Diagram for My Downloaded Movies App



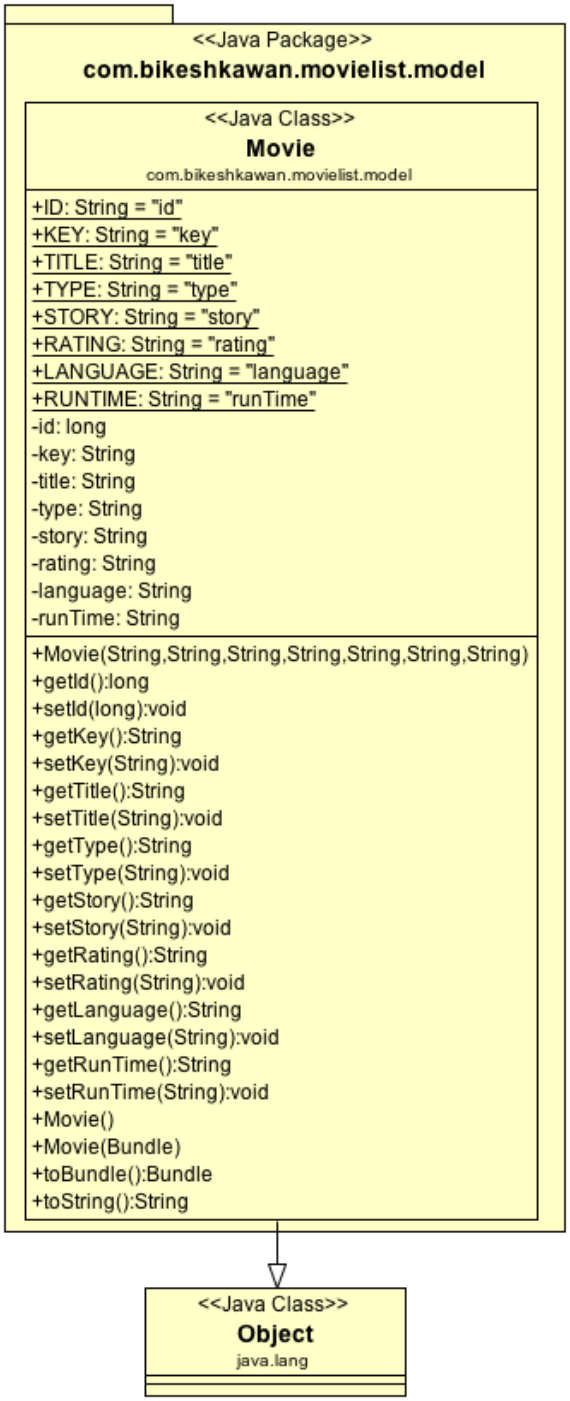
2.2 Class diagram for movielist package of main app



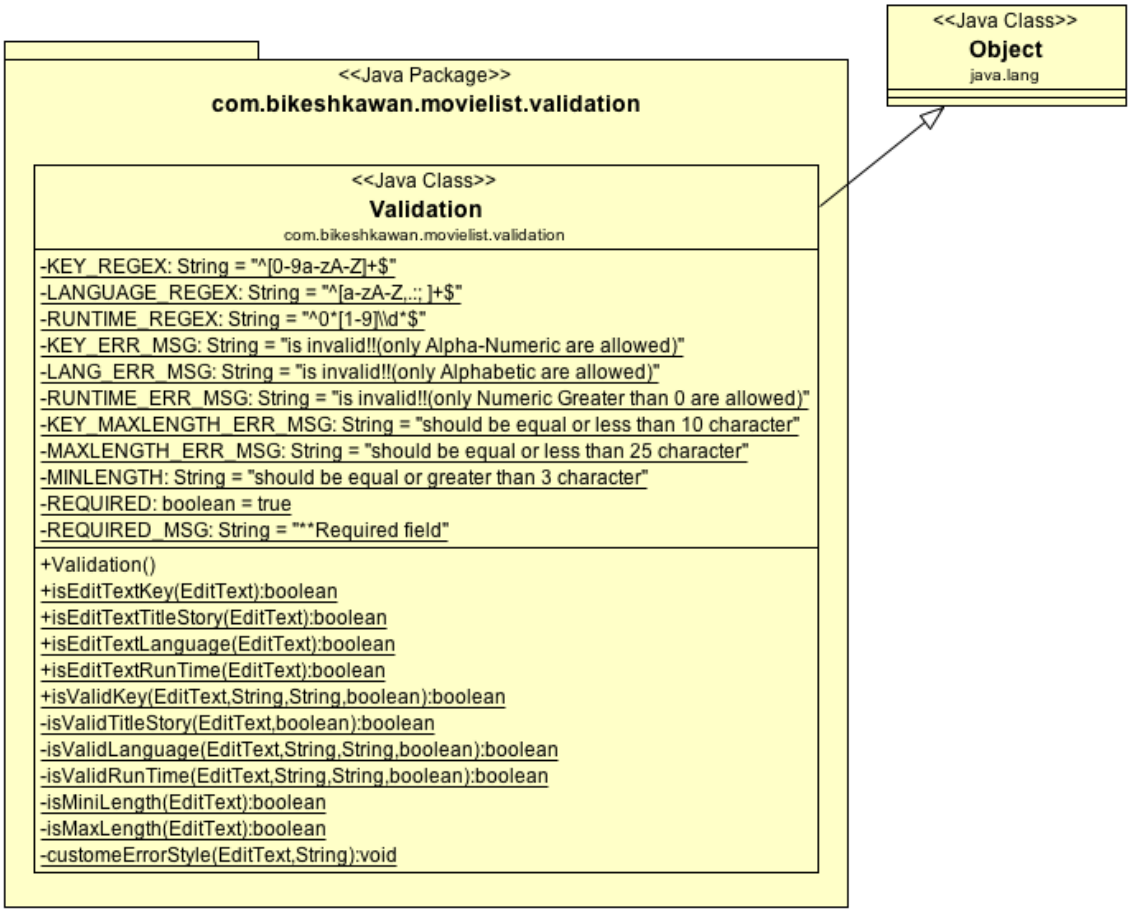
2.3 Class diagram for database package of main app



2.4 Class diagram package model of main app



2.5 Class diagram for validation package of main app



3 Story Boards

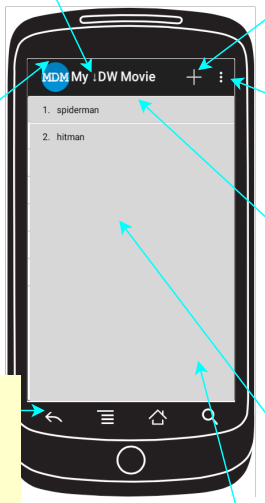
3.1 StoryBoard 1

App Name

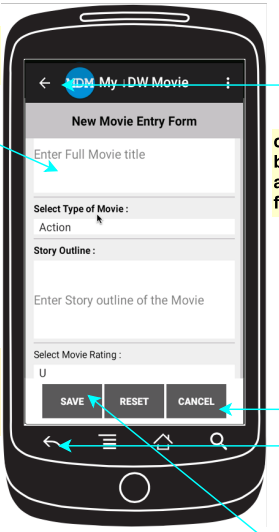
App Logo

Click to go to Screen 2 which is Add New movie entry Form.

Click to Select Menu Option in Home Screen

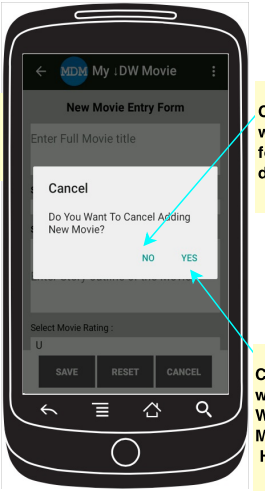


Screen 1



Screen 2

click on cancel and back button will open dialog box as in screen 6. A confirmation for quit adding movie details



Screen 6

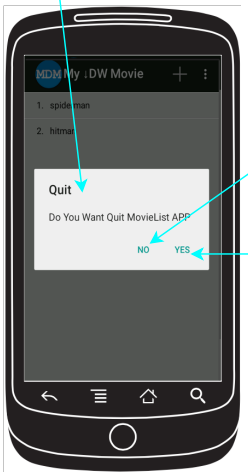
Click on 'NO' Button will stays in screen 2 for adding movie details

Click on 'YES' Button will go to Screen 1, Which contain list of Movies in Home Screen

click on Movie item will open screen 4, which displays details of movie item selected

Click to quit the app. It will open dialog box for quitting application as shown in screen 3

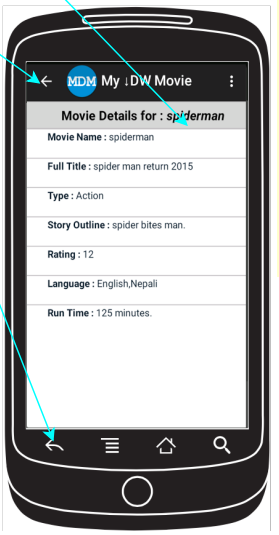
Click to go back to Main Screen ie Movie List Screen(Screen 1)



Screen 3

if, Clicked 'NO', app will go back to home screen ie screen 1

if, Clicked 'YES', app will close

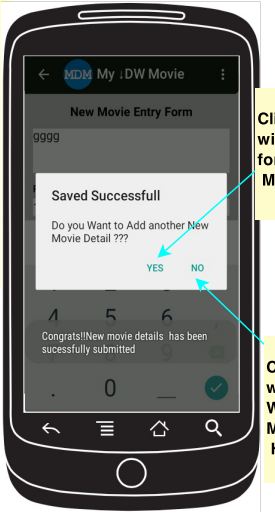


Screen 4

click save button to save movie . It will open screen 5, only if all the user input fields are valids and movie is unique and new.
It will save details into database with dialog box and Text confirmation for saved successfull

If, user input fields are invalid, It will show errors as in screen 7

if, Movie is already exists in database, it will open Screen 8, a dialog box for movie already Exists.

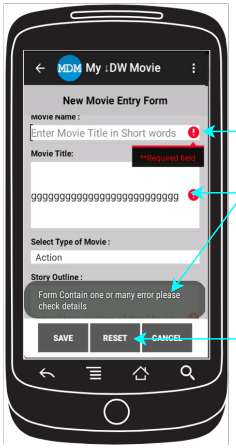


Screen 5

Click on 'YES' Button will go to Screen 2 for adding another Movie Details.

Click on 'NO' Button will go to Screen 1, Which contains list of Movies in Home Screen

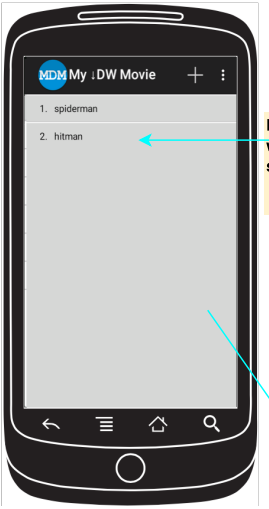
3.2 Story board 2



Screen 7

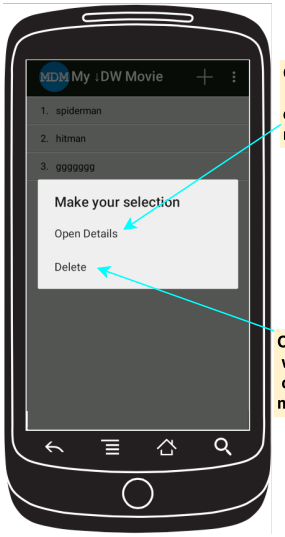
Display error message

Click on Reset Button will reset all the text Input fields



Screen 9

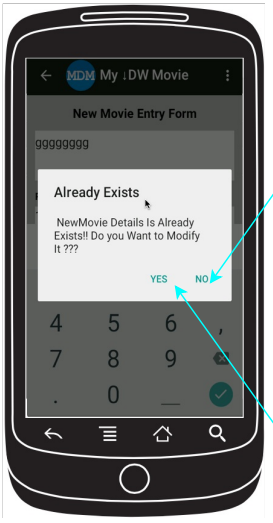
Long click on a " Movie Item" will open a dialog box for selection as shown in Screen 10 User can select either Open Details or Delete option



Screen 10

Click on Open Details will open Screen 4, which displays movie details of selected movie item

Click on Delete will open screen 11, which is a dialog box for delete confirmation for selected movie item

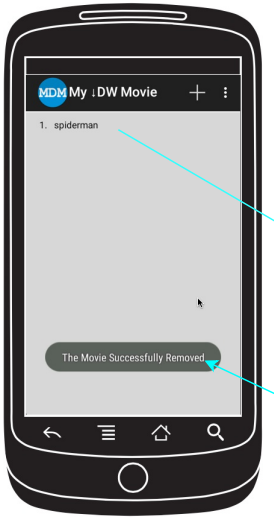


Screen 8

Click on 'No' Button will go to Screen9, which is home Screen and contains list of movies

Click on ' YES', Button will go move entry form i.e screen 2 so that user can edit movie details before saving

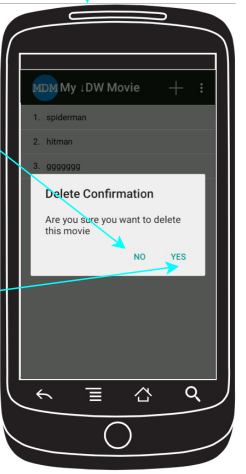
Click on "NO" Button will go back to screen 9, which is home screen. Selected Movie item will not be deleted from database



Screen 12

Click on "YES" Button will open to screen 12, which is home screen. Selected Movie item will be successfully deleted from Database and movie list on Screen 9 will upadate and new screen 12 will be displayed

Message displayed for user to let them know that selected movie item has been successfully deleted from database



Screen 11

3.3 Story board 3

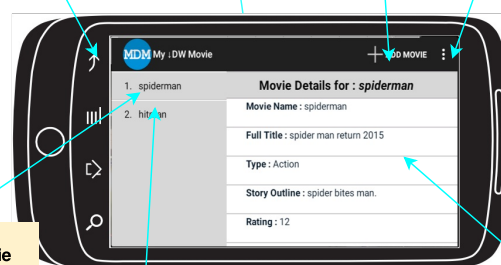
Screen 13

If, user rotate the device in horizontal mode, detail of first movie item will open in right side of the screen as shown in screen 14

Click on Add icon will open screen 2 i.e New movie Detail entry form in horizontal mode

Click to quit the app. It will open dialog box for quitting application as shown in screen 3

Click to Select Menu Option in Home Screen in horizontal mode



Taping movie item will open movie details of selected movie item in right hand side

Screen 14

Details of selected movie item from left handside

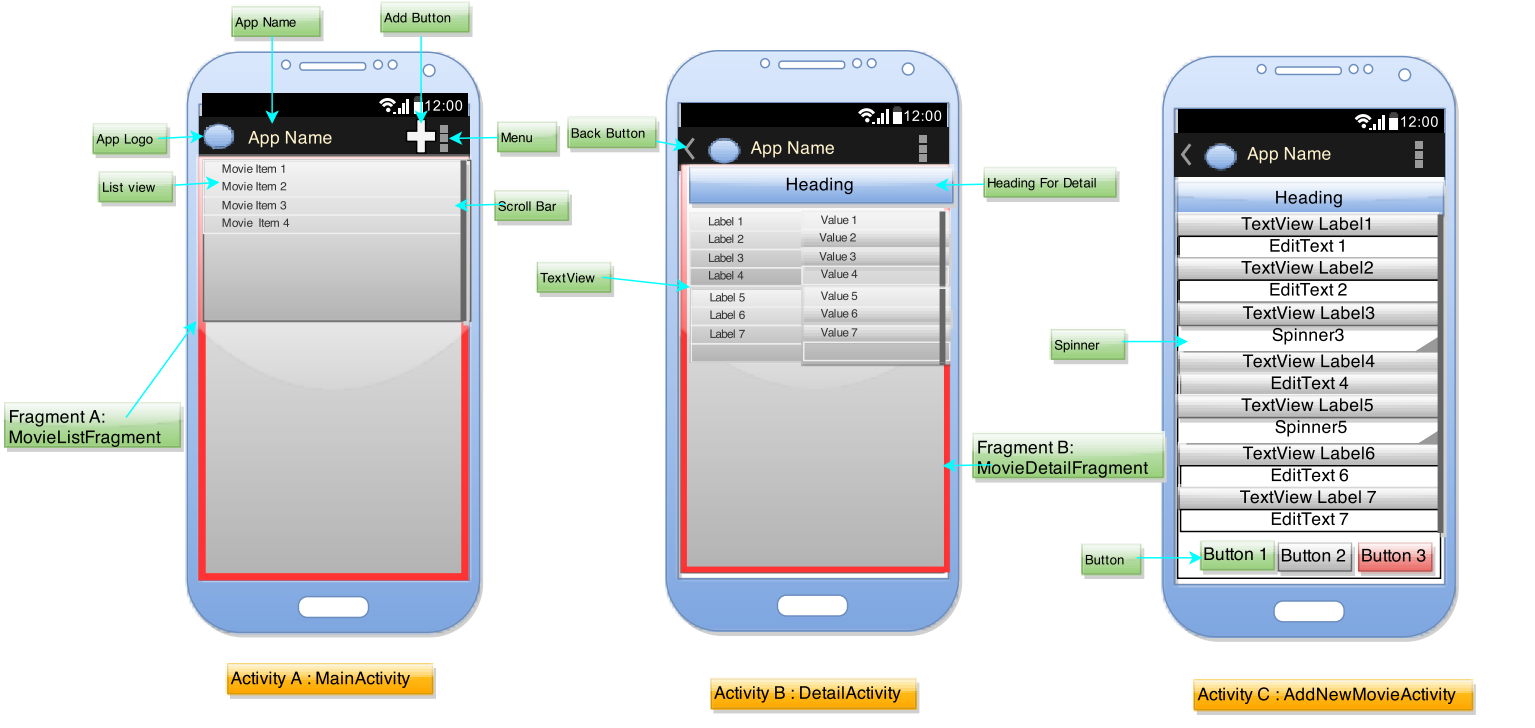
Long click on a " Movie Item" will display a dialog box for selection as shown in Screen 10 User can select either Open Details or Delete option.

If user select open details, details of selected movie item will be displayed in right hand side

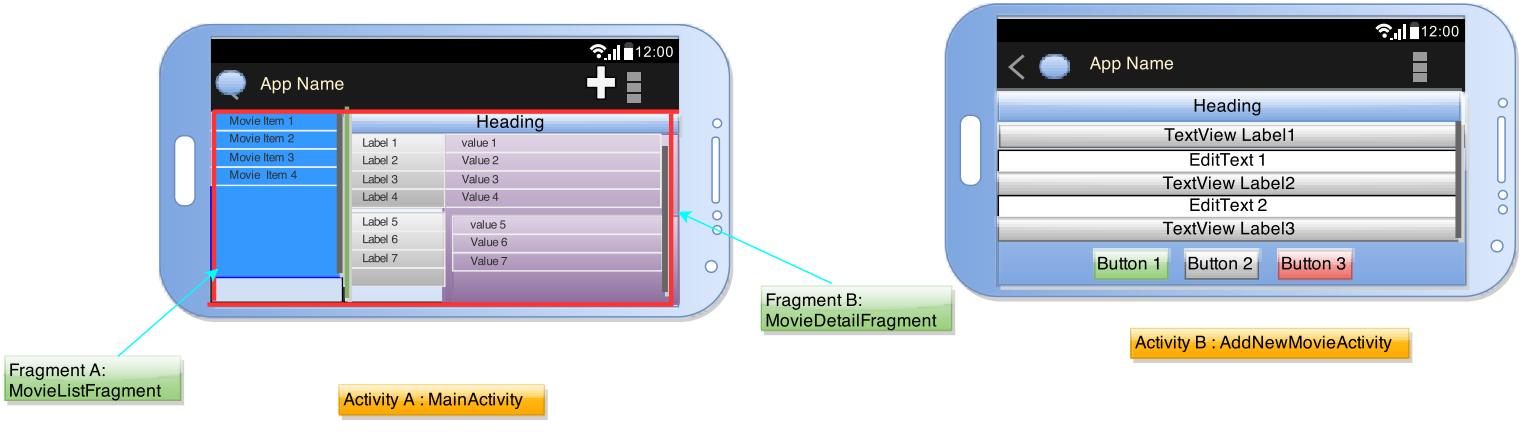
If user select delete , a dialog box will be displayed for delete confirmation and click on YES to delete the movie and click on NO will not delete movie from database.

4. Screen Layouts

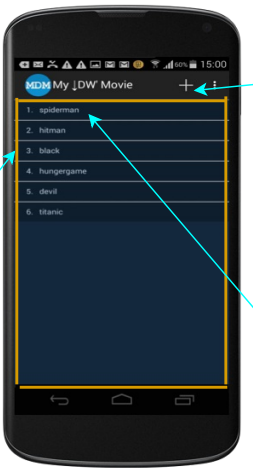
Screen Layout Design For Handset in Portrait Mode



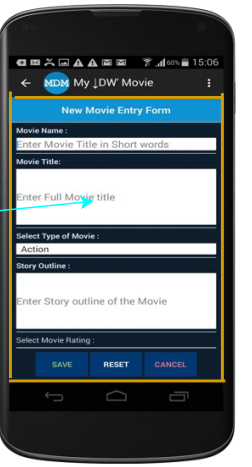
Screen Layout Design For Handset in Landscape Mode



Final Screen Layouts for Handset in Portrait Mode



Selecting Add Button Starts Activity C

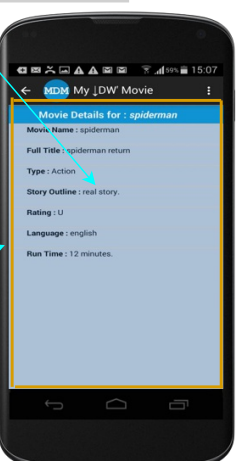


Fragment A :
MovieListFragment

Activity C : AddNewMovieActivity

Activity A : MainActivity
MainActivity Contains MovieListFragment

Selecting a Movie Item Starts Activity B

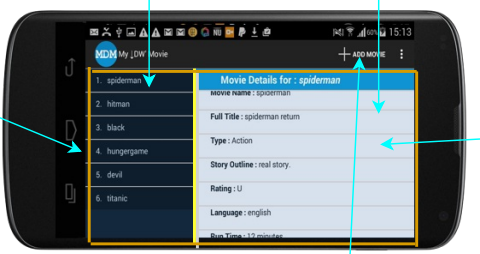


Fragment B :
MovieDetailFragment

Activity B: DetailActivity
DetailActivity Contains MovieDetailFragment

Final Screen Layouts for Handset in Landscape Mode

Selecting a Movie Item from Fragment A updates fragment B



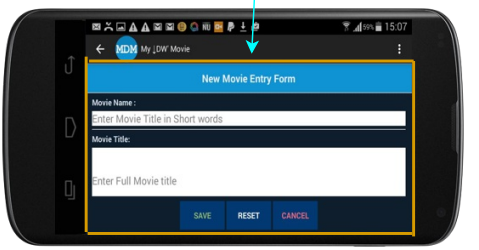
Fragment A : MovieListFragment

Fragment B : MovieDetailFragment

Activity A : MainActivity

MainActivity Contains MovieListFragment and MovieDetailFragment

Selecting Add Button in Activity A Starts Activity B



Activity B : AddNewMovieActivity

Optimization

Avoid creating unnecessary object

Creating an object means allocating memory, which is always more expensive than not allocating memory. [Android, 2015] Object instances have been created as less as possible while building app, as more objects will force a periodic garbage collection. I have followed two basic rules recommended by android.

1. Don't do work that you don't need to do
2. Don't allocate memory if you can avoid it.

Static methods and static variables have been used to avoid unnecessary object creation.
e.g., String string = new ("Hello World");// bad practice unnecessary object creation
e.g., String string = "Hello World"; // good practice

Optimizing Layout Hierarchies

Choice of layouts and inflating them into Activities also affect the user-interface loading speed. So, optimizing layout hierarchies are important to reduce inefficiencies and eliminate unnecessary nesting. (Meier, 2010) Wherever possible, I have avoided using too many views in layouts and they are inflated only when need as it takes time and resource to inflate.

Also, unnecessary and deep nesting have been avoided while creating xml layout file as they also take extra time to inflate. In order to do this, Linear Layouts have been replaced with a single Relative Layout, which will reduce the nesting level and improve the UI loading speed. (Nolan, Çinar and Truxall, 2014) Lint and Hierarchy Viewer helped me to identify deeply nested Liner Layouts.

Use of Static method Over Virtual method

It is good practice to make static method wherever possible, if accessing an object's fields is not required. Using static method is about 15%- 20% faster than virtual mode.(Android, 2015)Another benefits of static method is that object's state will not alter by call the method, which can be easily identify from method signature. In Validation class, static methods are used over virtual method. While validating the user input fields, AddNewMovieActivity does not need to access fields of Validation class.

Use of Static Final For Constants

It is good practice to declare constants static final for primitive type and string wherever possible. However, it cannot be applied to arbitrary reference types. All the codes are compiled into Dalvik byteCode. If a static final is used, the values of variable that are referenced will be accessed using instructions instead of a field lookup. I have used static final wherever possible. In data model i.e., Movie Class static final is used for its field reference.

In validation class, static finals have been used for error messages and regular expressions. Also, in data source class and database helper class, static finals have been used to represents database name, table name, column name etc. In Activities and fragments, most of the variables are also declared as static final. Compiled classes with static finals have better performance. Also, static final variables

can be directly called by other classes without instantiating it, which helps to avoid creating unnecessary object.

Avoid internal Getters/Setters

It is good to have getter and setter in public interface to follow common object-oriented programming practices, but within a class fields should be directly accessed. Virtual method calls are always expensive. Direct access to fields can increase the performance by seven times faster than invoking a getter. (Android, 2015) In my application, all the fields are directly accessed within classes. It helps to remove a line or two of bytecode from the generated classes.dx in the APK. (Nolan, Çinar and Truxall, 2014) Hence, performance of the app will be increased as fewer codes are complied.

Database Connection

Every activity in your application can open and close the database connection as needed. We can add as many calls to open database in every activity of our application. However, the connection object is always cached within the activity so we do not need to worry about calling the open method to many times. This will eliminate the possibility of database connection leaks. Database connection leaks can cause memory and performance issue. So, by adding the code to explicitly open and close the database when needed is good practice. The application will work well, work fast, and gives a reliable way to work with structured data. In lifecycle of Activities and fragments, wherever appropriate I have explicitly opened and closed the database connection in order to prevent from database connection leaks.

Avoid Using Floating-Point

Uses of floating-Points have been avoided while developing application. Floating-Points take long time for calculations and they are about two times slower than integer or long on Android devices. (Android, 2014)

Avoid Application Not Responding (ANR)

Even though, codes are optimized and have high performance, but still they might freeze for significant periods or take long time to process input action.[Android, 2015] Making network calls on the UI thread might cause compile error as well. We can keep our app responsive by using AsyncTask. I have used AsyncTaks when movie is added into database, updating movies and listing movies. This will enables proper and easy use of the UI thread. All the major operations are performed at background and results are published on the UI thread without need of manipulating threads. It is done in four steps: onPreExecute(), doInBackground(), onProgressUpdate(), onPostExecute().

Wherever possible I have avoided using inner class and anonymous inner class which access member of outer class .Dalvik VM features make this expensive.

Also, I have avoided checking length of array while using for loops. Before, deploying I have removed all the unnecessary and unused variables as well as it takes extra memory of the devices.

Portability

Supporting Different Screens

Android categorizes devices screen base on size and density. [Android, 2015] There are currently over 100 different screen sizes on Android i.e., includes Android phone, tablets etc. So, it is important that application developed will be work on devices with different screens.

To make app look good on various screen configuration there are two important things we need to take care. They are

1. Good Layout or structure for different screen sizes
2. Image work well with different screen resolution

I have provided alternative resources so that the app will run on different devices, which have different screen sizes and densities. One of the alternative resource layouts that I have used for a device is: orientation change. For this, I use land qualifier that will be used when a device is in landscape.

So, by default the layout/activity_main.xml file will be used when device is in portrait orientation. When a device is rotate to landscape mode, layout-land/activity_main.xml file will be used.

I have also provided alternative resource layouts for a large device.[Appendix A] For this, I have used large and land qualifier. So, when app runs on a large device in portrait orientation, by default the layout-large/activity_main.xml file will be used. And, when app runs on a large device in landscape orientation, layout-large-land/activity_main.xml file will be used.

I have also provided bitmap resources, which are properly scale to each of generalized density buckets. There are four generalized densities defined by android: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi). I have placed theses files in the appropriate drawable resource directory. [Appendix A]

In, current app, there are only two images: icon launcher and add button for action bar.

Now, anytime images are referenced, the system will select the appropriate bitmap base on the device's screen density. In this way, best graphical quality and performance of the app on different screen densities will be maintained.

Also, ScrollViews and ListViews are the easy solution to these problems when designing layouts for multiple screen sizes. There are huge list of screen sizes from phones to tablets but, most of the variation can be found in phone having the various screen heights. For this particular reason, ScrollViews and ListViews work well and that I have implemented in my application. If the screen is large enough user can see details and list in one go, but if, device is small they can scroll or if content is more they can scroll and view it. However, it is best if we can design our layout that works well across all screens without ScrollViews.

Wherever possible, I have used "wrap_content" and "match_parent" for the width and height of some view component. This will help the layout to be flexible and adapts to different screen sizes and orientation. Also, I have used RelativeLayout in most of the layout file such as movie_list.xml, movie_list_fragment.xml etc. This will allow position of the child views relative to each other.

Use of Absolute Layout has been avoided as it enforces the use of fixed position to layout its child views. Also, AblsoluteLayout is deprecated.

When designing layouts for the app, I have avoided using absolute pixels to define distance and sizes, as different devices have different screen with different pixel densities [Android, 2015].Therefore, I have used dp(density-independent pixel) to define layout dimensions i.e., layout size and sp (scale-independent pixel) to define text size.

Supporting different Versions of Android Platform

The dashboard for android platform versions is update regularly. [Appendix B]
It is good practice to target app to the latest version and, to support more than 90% of active devices. The detail about the app and versions level of android is described at AndroidManifest.xml. So, first thing, I have to specify the lowest API level with which the app is compatible and the highest API level against which the app is designed and tested.

In order to specify, I gave '8' for minSdkVersion and '22' for targetSdkVersion attributes for the <use-sdk element.

Support Library

I have used Android Support Library packages so that the app will run in older devices as well. The current app developed uses fragments, which provide dynamic and multi-pane user interface. Also, action bar has been used. But, fragment class will not work in API level lower than 11. I have used v4 support Library so that fragment class is also compatible with Android 2.1(API Level 7) and higher version. [Appendix A] This will enable application to provide layouts that adjust between small and large –screen devices. Now, the app will run in API Level 8 and higher version.[Appendix D] [Appendix E]

I have also used v7 appcompat library, which provides support for the Action Bar user interface design pattern. This is also compatible with Android 2.1(API Level 7) and higher version and also includes the Fragment APIs. Now, implementation of the action bar is possible in API Level 7 and above. [Appendix D] [Appendix E]

References

Android, 2015. Keeping Your App Responsive | Android Developers. [online]

Developer.android.com. Available at: <http://developer.android.com/training/articles/perf-anr.html> [Accessed 5 Jun. 2015].

Android, 2015. Performance Tips | Android Developers. [online] Developer.android.com.

Available at: <http://developer.android.com/training/articles/perf-

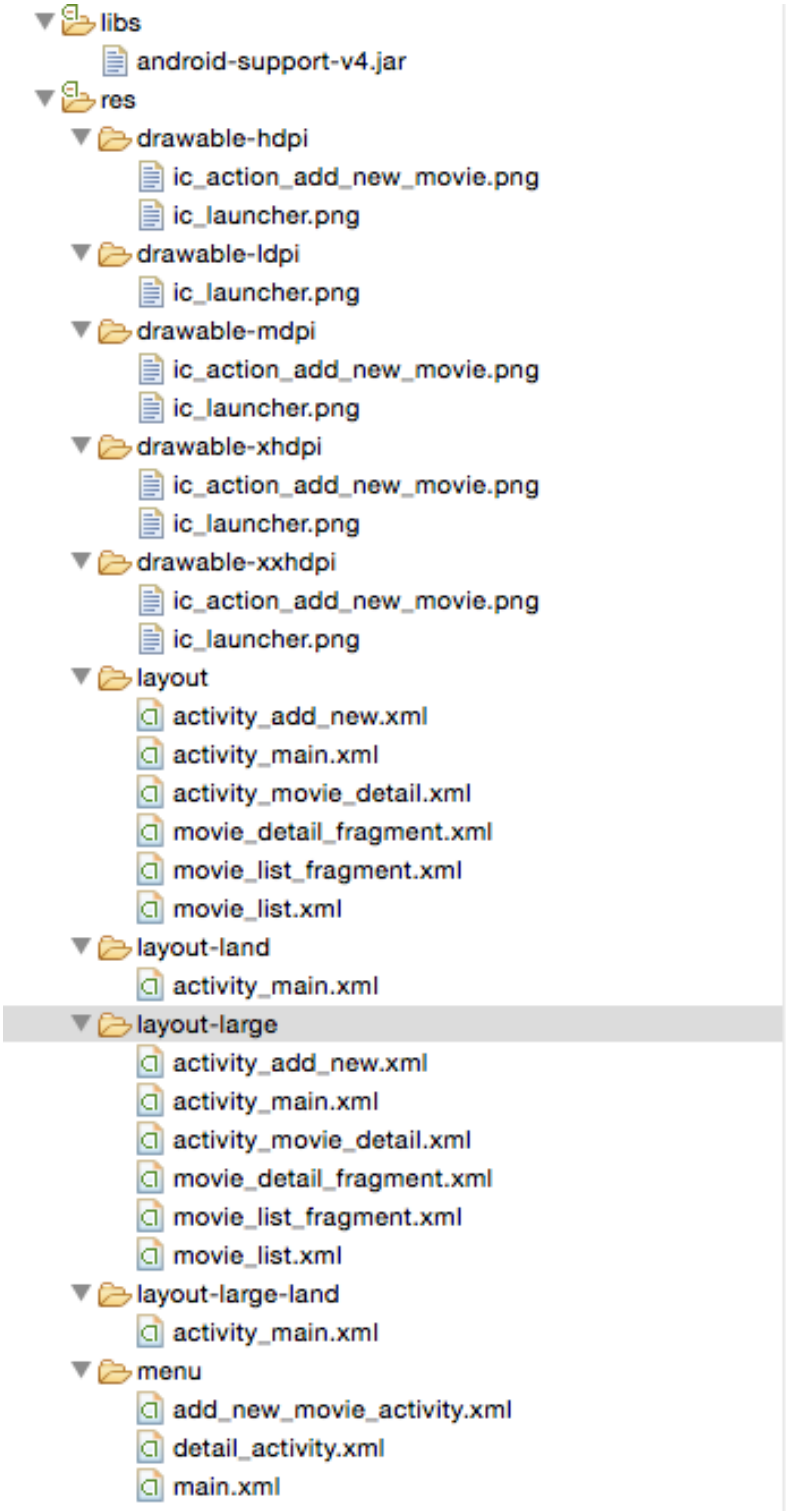
tips.html#ObjectCreation> [Accessed 1 Jun. 2015].

Meier, R., 2010. Professional Android 2 application development. Indianapolis, Ind.: Wiley.

Nolan, G., CÌ§inar, O. and Truxall, D., 2014. Android best practices. [New York]: Apress.

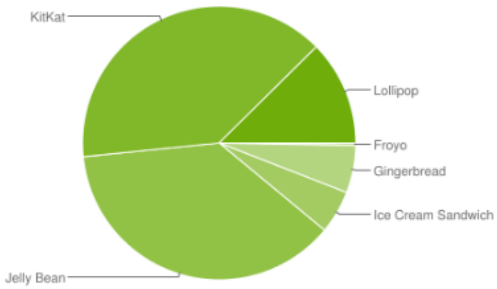
Appendices

Appendix A



Appendix B

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



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

Appendix C

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="22" />

<application
```

Appdendix D

Fig: Action bar and Fragment working in API Level 8

Appendix E

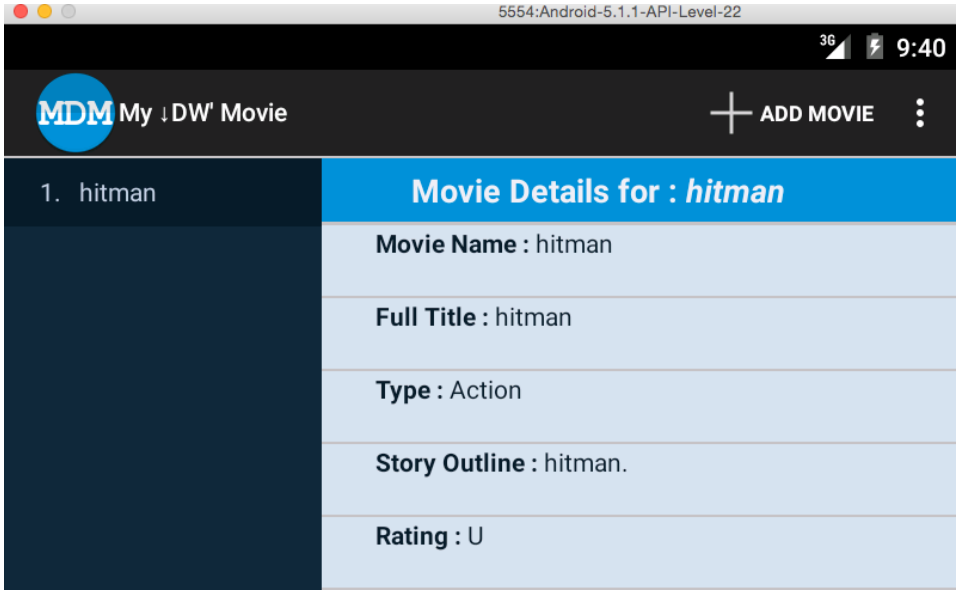


Fig: Action bar and Fragment working in API Level greater than 8