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 than 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 paramenters, 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;
```

Now in onCreate Method we initialize it .

```
              moviesList = new ArrayList<Movie>();
```

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

```
              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.setListAdapter(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 contruct 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.

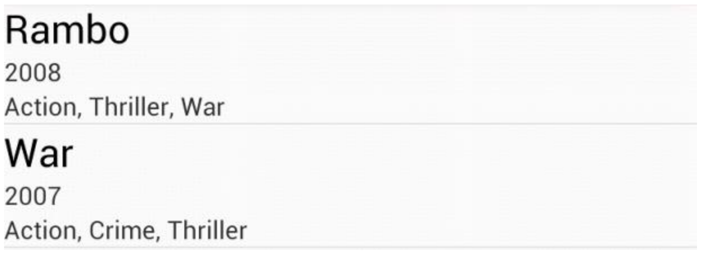
Rambo
2008
Action, Thriller, War
War
2007
Action, Crime, Thriller

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 than 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();
	}
```