2016

Session 3 - Exercises

V1.0

JUAN RONDON

Table of Contents

Part 1 – ListView

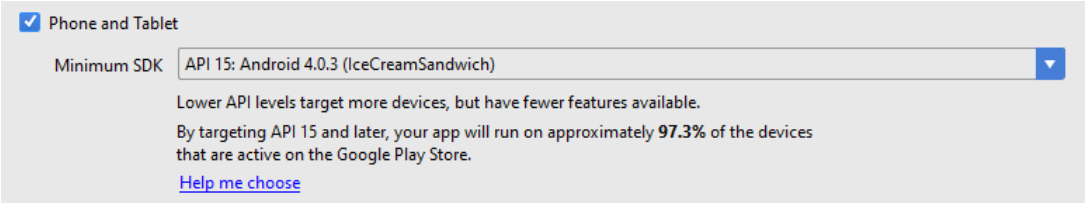
Exercise 1 – Simple List View

1. Open Android Studio and create a new Android Application

2. From the configure window give your application the name of GamesApp:

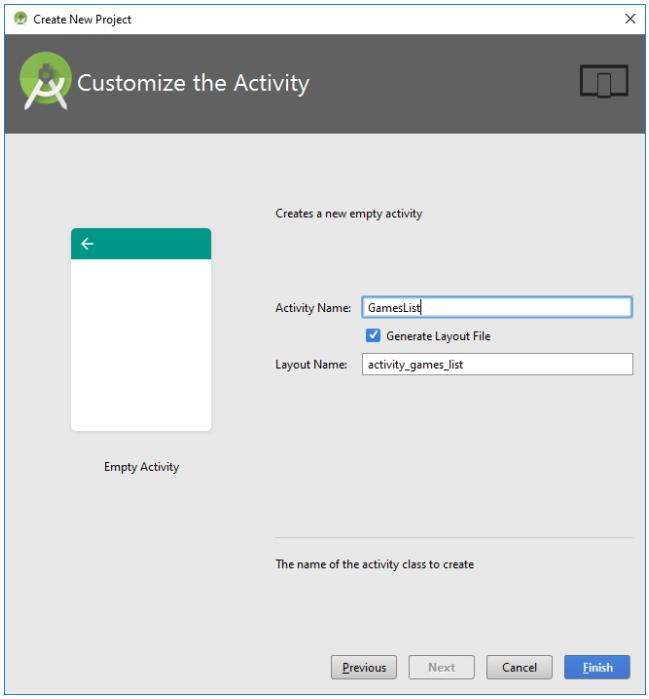
   Note: Use any project location but save the project with the name of **GamesApp.**

3. In the form factors window select:



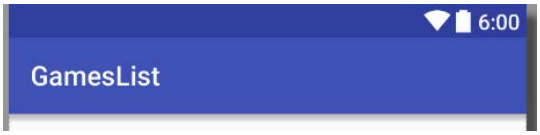
4. In the next window make sure to select an **Empty Activity.**

5. Name your activity **GamesList.**

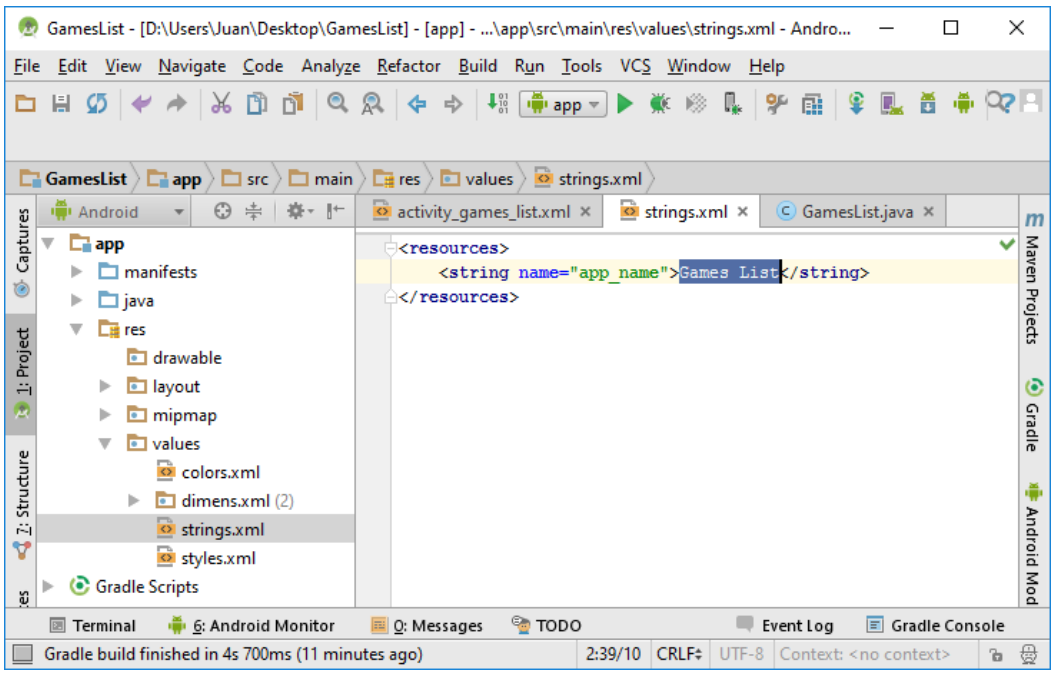


6. Remove the existing Hello World! **TextView** from **activity_games_list.xml**

7. Next you will need to modify the text from the action bar so it will display **Games List** instead of **GamesList**

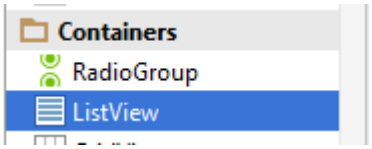


8. Open Strings.xml from res → values folder and make sure to add the whitespace between the words Games and List

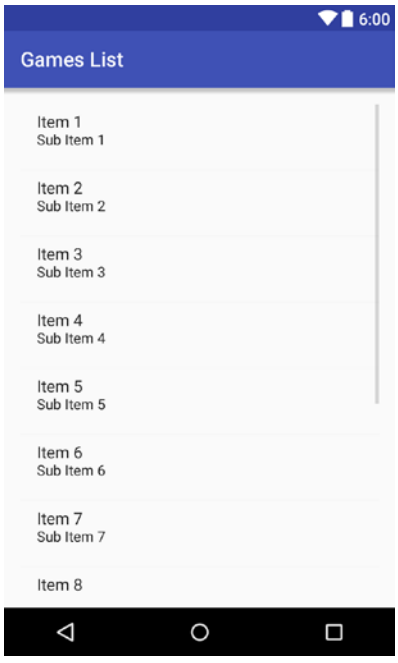


9. Return to **activity_games_list.xml** and see the changes.

10. From the containers add a **ListView** widget to your activity layout and give it the **id** of **listViewGames**



11. Your layout for the activity should look like the following image.



12. Open **GamesList.java** file and create the following array with the data required to display in the ListView

```
package androidcourse.gameslist;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class GamesList extends AppCompatActivity {

    // Example data, using an in-memory String array...
    private String[] mGamesCollection = new String[]{
            "Call of Duty: Black Ops 3",
            "Fallout 4",
            "Far Cry Primal",
            "Just Cause 3",
            "Life is Strange",
            "Mad Max",
            "Metal Gear Solid V: The Phantom Pain",
            "Need for Speed",
            "Project CARS",
            "Rise of the Tomb Raider",
            "Star Wars Battlefront",
            "The Witcher 3: Wild Hunt",
            "Tom Clancy's Rainbow Six Siege",
            "Wolfenstein: The Old Blood"
    };

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

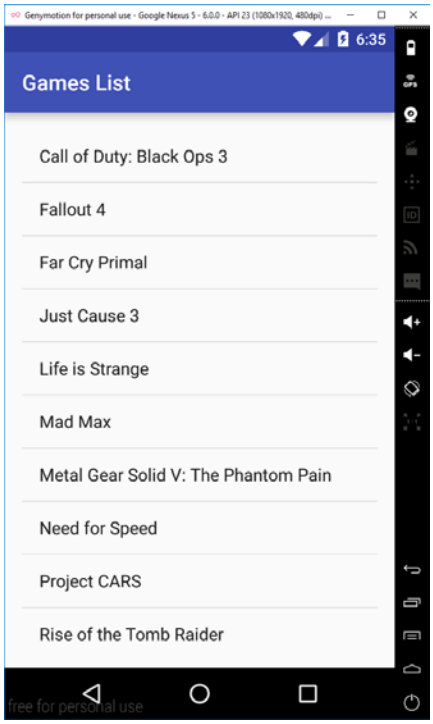
13. The next step would be to connect the **mGamesCollection** data array with the **ListView** from the activity. In order to do that, create the following code inside of **onCreate** method.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_games_list);
        // Get the ListView from our xml layout
        ListView mListViewGames = (ListView) findViewById(R.id.listViewGames);

        // Create an Adapter to bind our data to a row template (View), and populate the
ListView with the row template
        ArrayAdapter<String> mArrayAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, mGamesCollection);
        // Context -> Activity (this class is an Activity)
        // int Resource -> The id of a View (row template)
        // NOTE: android.R.xxx <- instead of R.xxx, android contains existing resources we can
use :)
        // String[] -> The collection (our data)

        // Set the ListView's Adapter
        mListViewGames.setAdapter(mArrayAdapter);
    }
}
```

14. Run your application in the emulator/device.

15. At this point you should see the list of games as a **ListView** and you should be able to scroll through them.

Continues in next page



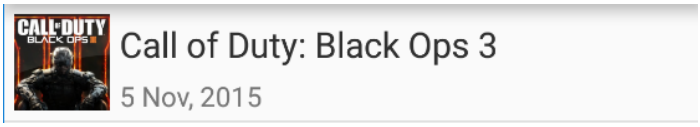
16. Now you will be adding a listener in order to do something every time the user clicks on a game from the list. Add the following code after you set the listView's adapter from step 13.

```
// Set an OnItemClickListener for the ListView, and use the default anonymous class...
mListViewGames.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        // When an item (row) is clicked, lets display a Toast
        // First lets get the String of the item using the position passed to this callback
        String gameName = mGamesCollection[position];   // NOTE: 0 index based

        Toast.makeText(getApplicationContext(), gameName, Toast.LENGTH_SHORT).show();
        // Context -> rather than using MainActivity.this (to refer to the context/activity)
        //            we can use getApplicationContext()
    }
});
```

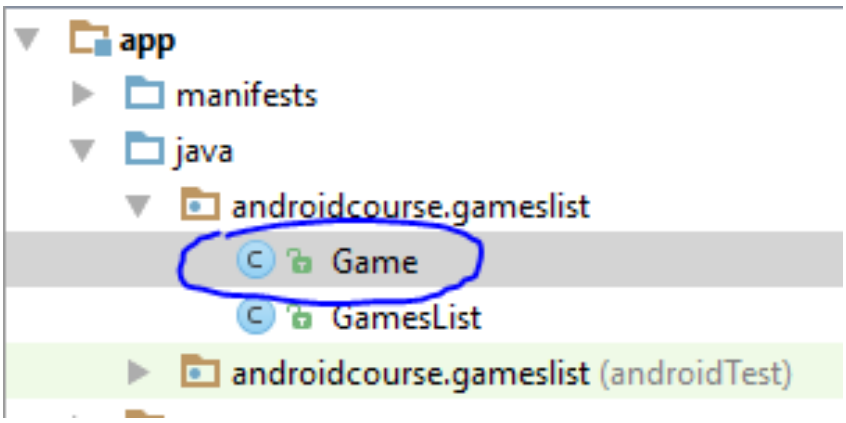
Now each time that you click on a game, your app will display a Toast with the name of the selected game.

Exercise 2 – Custom List View

1. So far the application looks alright but what if you want to display an image for each of the games and maybe the release date too…



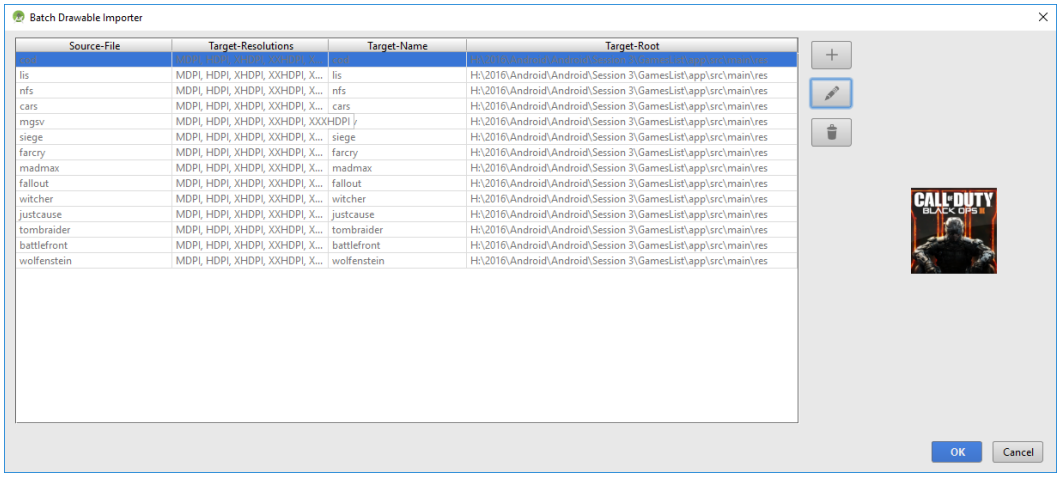
2. In this case we need to create a custom adapter that will be able to handle a more complex layout for each element of the ListView.
3. First we will need to create a Game class that will contain data values for the game name, the image and the release date. Create a new class named **Game** inside the java folder:



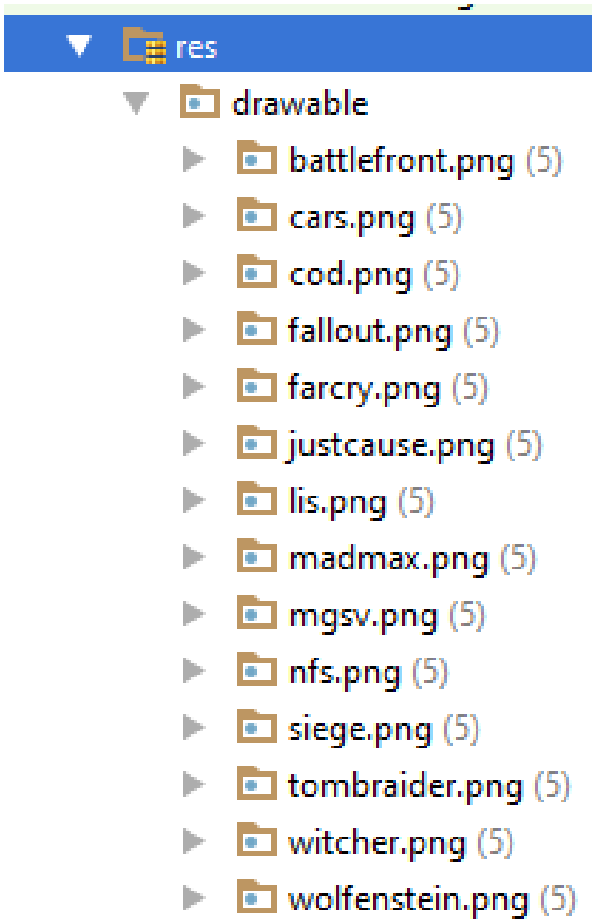
Add the following code to the class:

```
package androidcourse.gameslist;

public class Game {
    private String mName;
    private String mCoverImageName;
    private String mReleaseDate;

    public Game(String name, String coverImageName, String releaseDate) {
        mName = name;
        mCoverImageName = coverImageName;
        mReleaseDate = releaseDate;
    }

    public String getName() {
        return mName;
    }
    public String getCoverImageName() {
        return mCoverImageName;
    }

    public String getReleaseDate() {
        return mReleaseDate;
    }
}
```

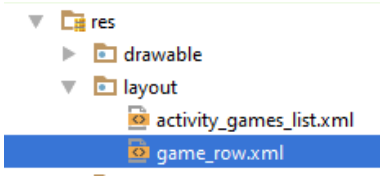
4. Now you will be adding all the images required for all the games.
Make sure to use the Drawable Importer plugin in order to add all the images to the correct folders (hdpi, mdpi, xhdpi, etc).



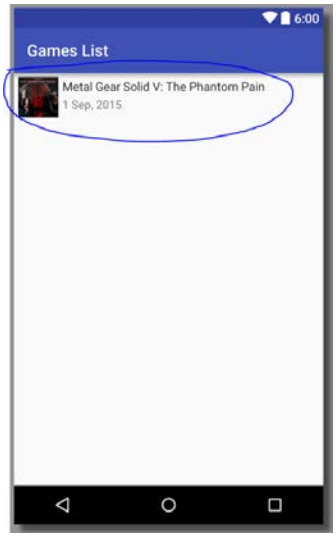
5. Once the images are added, the **Drawable** folder should contain different resolutions for all the game images.



6. The next step would be to create the custom layout for the ListView. The layout is an xml file with a template for a single row of the ListView.
   Create a new xml file named **game_row.xml** inside the **layout** resources folder. (make sure to use lowercase for the layout name otherwise it won't work).

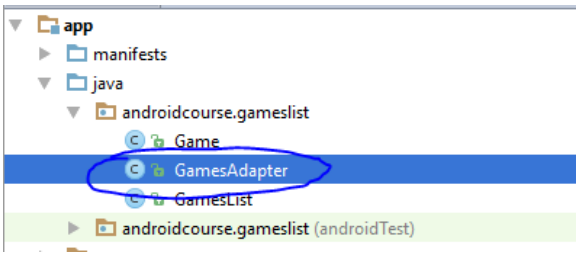



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout height="wrap content">

    <ImageView
        android:id="@+id/imageViewCover"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_margin="5dp"
        android:src="@drawable/mgsv" />

    <TextView
        android:id="@+id/textViewName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="8dp"
        android:layout_toEndOf="@+id/imageViewCover"
        android:layout_toRightOf="@+id/imageViewCover"
        android:text="Metal Gear Solid V: The Phantom Pain"
        android:textColor="#303030"
        android:textSize="15sp" />

    <TextView
        android:id="@+id/textViewReleaseDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textViewName"
        android:layout_marginTop="5dp"
        android:layout_toEndOf="@+id/imageViewCover"
        android:layout_toRightOf="@+id/imageViewCover"
        android:text="1 Sep, 2015"
        android:textColor="#777777" />
</RelativeLayout>
```



Note: Don't worry about hardcoding the image and text for the name and the release date, it will be replaced later on with all the values from your games collection data. This is just for you to preview the look and feel of the row of your ListView.

7. Now you will need to create a custom adapter that will work with the proposed ListView, the adapter will map each of the Games to a new row of the ListView using the layout created in the previous step.
    a. Create a class named **GamesAdapter** inside java folder.



    b. Add the following code to this class:

```
package androidcourse.gameslist;

import android.content.Context;
import android.nfc.cardemulation.CardEmulation;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class GamesAdapter extends ArrayAdapter<Game> {
    private Context mContext;
    private int mLayoutResourceId;
    private Game[] mGames;

    public GamesAdapter(Context context, int resource, Game[] objects) {
        super(context, resource, objects);
        mContext = context;
        mLayoutResourceId = resource;
        mGames = objects;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View row = convertView; // lets name convertView row, as its easier to understand

        // Get the LayoutInflator from Activity/Context
        // Instantiates a layout XML file into its corresponding View objects
        LayoutInflater inflater = LayoutInflater.from(mContext);
        // Inflate the row template with the xml layout passed to this Adapter
        row = inflater.inflate(mLayoutResourceId, parent, false);
        // ensure attachToRoot is false otherwise you may get an exception:
        // java.lang.UnsupportedOperationException: addView(View, LayoutParams)
        //is not supported in AdapterView

        // Get the View elements in the row template

        // NOTE: we must use row.findViewById() as we are in a class that is not an Activity
        ImageView coverImage = (ImageView) row.findViewById(R.id.imageViewCover);
        TextView gameName = (TextView) row.findViewById(R.id.textViewName);
        TextView releaseDate = (TextView) row.findViewById(R.id.textViewReleaseDate);

        // Get the Game object for the current position this method is requesting
        Game game = mGames[position];

        // Get the id of the drawable (image) for the game
        int resId = mContext.getResources().getIdentifier(game.getCoverImageName(), "drawable", mContext.getPackageName());
        // name -> The file name of the resource (no extensions required!)
        // defType -> The type of the resource
        // defPackage -> The package where the resource lives

        // Now lets set the View elements with the values for the game
        coverImage.setImageResource(resId);
        gameName.setText(game.getName());
        releaseDate.setText(game.getReleaseDate());

        // Finally we return the row (View) so the ListView can display it :)
        return row;
    }
}
```

8. The last step would be to create a collection of game objects.
   Open GamesList class and replace the game data array with the following code:

```
// Example data, using an in-memory String array...
private Game[] mGamesCollection = new Game[] {
        new Game("Call of Duty: Black Ops 3", "cod", "5 Nov, 2015"),
        new Game("Fallout 4", "fallout", "9 Nov, 2015"),
        new Game("Far Cry Primal", "farcry", "23 Feb, 2016"),
        new Game("Just Cause 3", "justcause", "1 Dec, 2015"),
        new Game("Life is Strange", "lis", "30 Jan, 2015"),
        new Game("Mad Max", "madmax", "1 Sep, 2015"),
        new Game("Metal Gear Solid V: The Phantom Pain", "mgsv", "1 Sep, 2015"),
        new Game("Need for Speed", "nfs", "3 Nov, 2015"),
        new Game("Project CARS", "cars", "6 May, 2015"),
        new Game("Rise of the Tomb Raider", "tombraider", "10 Nov, 2015"),
        new Game("Star Wars Battlefront", "battlefront", "17 Nov, 2015"),
        new Game("The Witcher 3: Wild Hunt", "witcher", "18 May, 2015"),
        new Game("Tom Clancy's Rainbow Six Siege", "siege", "1 Dec, 2015"),
        new Game("Wolfenstein: The Old Blood", "wolfenstein", "5 May, 2015")
};
```

Note: notice that now the array contains Game objects instead of simple Strings, each object contains the name of the game, the image name and the release date.

9. Make the required changes to **onCreate** method so it will contain the following code:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_games_list);
    // Get the ListView from our xml layout
    ListView mListViewGames = (ListView) findViewById(R.id.listViewGames);

    // Create an Adapter to bind our data to a row template (View), and populate the
    //ListView with the row template
    GamesAdapter adapter = new GamesAdapter(this, R.layout.game_row, mGamesCollection);
    // Context -> Activity (this class is an Activity)
    // int Resource -> The id of a View (row template)      we use our custom View game_row
    // Game[] -> The collection (our data)

    // Set the ListView's Adapter
    mListViewGames.setAdapter(adapter);

    // Set an OnItemClickListener for the ListView, and use the default anonymous class...
    mListViewGames.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            // When an item (row) is clicked, lets display a Toast
            // First lets get the String of the item using the position passed to this callback
            String gameName = mGamesCollection[position].getName();   // NOTE: 0 index based

            Toast.makeText(getApplicationContext(), gameName, Toast.LENGTH_SHORT).show();
            // Context -> rather than using MainActivity.this (to refer to the context/activity)
            //            we can use getApplicationContext()
        }
    });
}
```

10. Run the application on the emulator/device and notice the rendered list of games.

Each time that you click on a game, your app will display a Toast with the name of the selected game.

Part 2

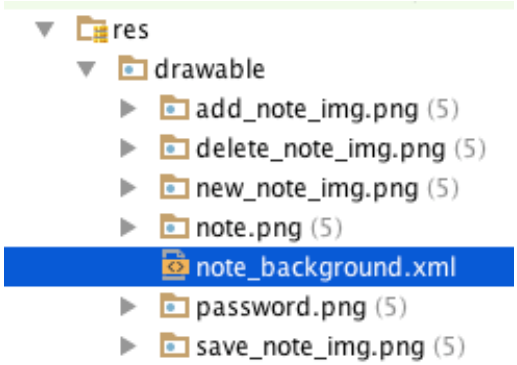
Android Note taking application (Including ViewHolder pattern)

1. Last session you created Note class and added all the required logic for it. This time we will be creating some hardcoded notes and we will be displaying them in **NotesList** Activity.

2. As a template for each of the notes we want to create the following xml layout:

3. Add the required new images into **Drawable** folder using the **Drawable** importer (same as steps 4-5 from Part 1). **Note**: leave the same names because they will be used later from the layout.

4. Create a new Drawable item named **note_background.xml.**



5. Add the following code to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#99ffffff" />
    <corners android:radius="5dp" />
    <padding
        android:bottom="10dp"
        android:left="10dp"
        android:right="10dp"
        android:top="10dp" />
</shape>
```

Note: The idea behind this drawable xml is to provide a custom rounded corner and background to each of our notes.

6. Create a new resource file inside layout folder named **note_row.xml** (Same steps as previous exercise). Add the following code to it:

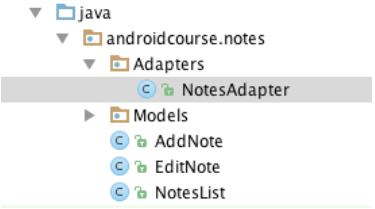
Continues in next page

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="3dp"
    android:background="@drawable/note_background"
    android:orientation="vertical"
    android:paddingBottom="5dp"
    android:paddingRight="10dp"
    android:paddingTop="5dp">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/lastModified"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:src="@drawable/note" />

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_toEndOf="@+id/icon"
        android:layout_toRightOf="@+id/icon"
        android:text="Note Title"
        android:textColor="#030303"
        android:textSize="18sp" />

    <ImageView
        android:id="@+id/pwdImg"
        android:layout_width="18dp"
        android:layout_height="18dp"
        android:layout_alignBottom="@+id/title"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:src="@drawable/password" />

    <View
        android:id="@+id/line"
        android:layout_width="fill_parent"
        android:layout_height="1dip"
        android:layout_below="@+id/title"
        android:layout_marginTop="2dp"
        android:layout_toEndOf="@+id/icon"
        android:layout_toRightOf="@+id/icon"
        android:background="#777777" />

    <TextView
        android:id="@+id/lastModified"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/line"
        android:layout_alignRight="@+id/line"
        android:layout_below="@+id/line"
        android:text="Last Modified"
        android:textColor="#555555"
        android:textSize="10sp" />

</RelativeLayout>
```

7. After this step your design view for **note_row.xml** should look like the template from step 2.

8. Now you will need to create a custom adapter that will work with the proposed ListView, the adapter will map each of the Notes to a new row of the ListView using the layout created in the previous step.
    a. Create a package/folder for the adapters and name it **Adapters**
    b. Create a class named **NotesAdapter** inside **Adapters** folder.

Sample image in next page



c. Add the following code to this class:

```
package androidcourse.notes.Adapters;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;

import androidcourse.notes.Models.Note;
import androidcourse.notes.R;

public class NotesAdapter extends ArrayAdapter<Note> {

    private Context mContext;
    private ArrayList<Note> mNotelist;
    private int mLayoutResourceId;

    private static class ViewHolder {
        TextView title;
        TextView date;
        ImageView img;
        ImageView pwd;
    }

    public NotesAdapter(Context context, int layoutResourceId, ArrayList<Note> notelist) {
        super(context, layoutResourceId, notelist);
        mContext = context;
        mNotelist = notelist;
        mLayoutResourceId = layoutResourceId;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ViewHolder viewHolder;
        View rowView = convertView;

        /* There is no view at this position, we create a new one.
       In this case by inflating an xml layout */

        if (rowView == null) {
            LayoutInflater inflater = LayoutInflater.from(mContext);
            rowView = inflater.inflate(mLayoutResourceId, parent, false);
            // configure view holder
            viewHolder = new ViewHolder();
            viewHolder.title = (TextView) rowView.findViewById(R.id.title);
            viewHolder.date = (TextView) rowView.findViewById(R.id.lastModified);
            viewHolder.img = (ImageView) rowView.findViewById(R.id.icon);
            viewHolder.pwd = (ImageView) rowView.findViewById(R.id.pwdImg);
            rowView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolder) rowView.getTag();
        }
        //fill data
        viewHolder.title.setText(mNotelist.get(position).getmTitle());
        viewHolder.date.setText(mNotelist.get(position).dateFormatted());
        if (mNotelist.get(position).getmPassword() != null) {
            viewHolder.pwd.setImageResource(R.drawable.password);
        } else {
            viewHolder.pwd.setImageDrawable(null);
        }
        return rowView;
    }
}
```

9. The last step would be to create an object from the adapter and assign it to the ListView.
   Open **NotesList** Activity and create a few hardcoded notes:

```
private ArrayList<Note> notes = new ArrayList<Note>() {{

    add(new Note("Note 1", "This is an example of my first note"));
    add(new Note("Note 2", "This is an example of my second note"));
    add(new Note("Note 3", "This is an example of my third note", "123"));
    add(new Note("Note 4", "This is an example of my fourth note"));
    add(new Note("Note 5", "This is an example of my fifth note", "123"));
}};
```

10. Make the required changes to **onCreate** method so it will contain the following code:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_notes_list);
    final ListView notesListView = (ListView) findViewById(R.id.listView);
    NotesAdapter adapter = new NotesAdapter(this, R.layout.note_row, notes);
    notesListView.setAdapter(adapter);

    notesListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            Toast.makeText(NotesList.this, notes.get(position).getmTitle().toString(),
Toast.LENGTH_SHORT).show();
        }
    });
}
```

11. Run the application on the emulator/device and notice the rendered list of notes.
    Each time that you click on a game, your app will display a Toast with the title of the selected note.