Queens University Belfast

Android Studio

CSC3054 / CSC7054

GridView Exercises

Book1

Android `GridView`

This layout shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a `ListAdapter`.

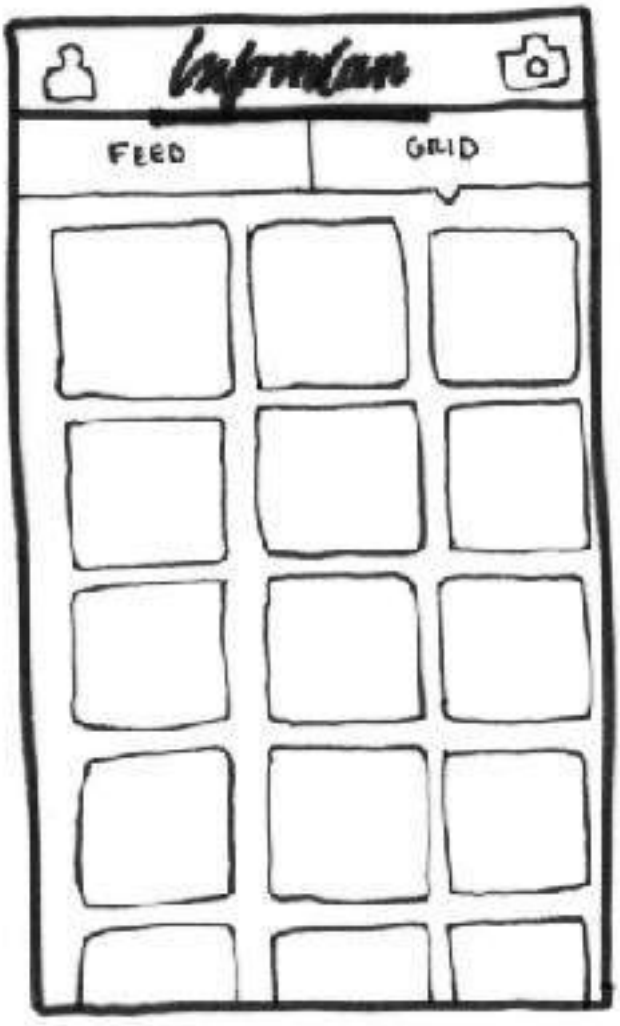

FIGURE 1 GRIDVIEW LAYOUT

An `adapter` is a controller object that sits between the `View` component and the data set containing all the data that the `View` component should display. An `adapter` can be used to supply the data to: `Spinner, ListView, GridView` etc.

The `ListView` and `GridView` are subclasses of `AdapterView` and binding them to an `Adapter` can populate these `View` components. The `Adapter` retrieves data from an external source and creates a `View` that represents each data entry.

Gridview Attributes

Following are the important attributes specific to `GridView`

Attribute	Description
`android:id`	This is the ID, which uniquely identifies the layout.
`android:columnWidth`	This specifies the fixed width for each column. This could be in `px, dp, sp`, in, or mm.
`android:gravity`	Specifies the `gravity` within each cell. Possible values are `top, bottom, left, right, center, center_vertical, center_horizontal` etc.
`android:horizontalSpacing`	Defines the default horizontal spacing between columns. This could be in `px, dp, sp, in`, or `mm`.
`android:numColumns`	Defines how many columns to show. May be an integer value, such as `"100"` or `auto_fit` which means display as many columns as possible to fill the available space.
`android:stretchMode`	Defines how columns should stretch to fill the available empty space, if any. This must be either of the values – • `none`: Stretching is disabled. • `spacingWidth`: The spacing between each column is stretched. • `columnWidth`: Each column is stretched equally. • `spacingWidthUniform`: The spacing between each column is uniformly stretched.
`android:verticalSpacing`	Defines the default vertical spacing between rows. This could be in `px, dp, sp, in`, or `mm`.

Exercise 1 – Create the layout

In this tutorial, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

Final Output

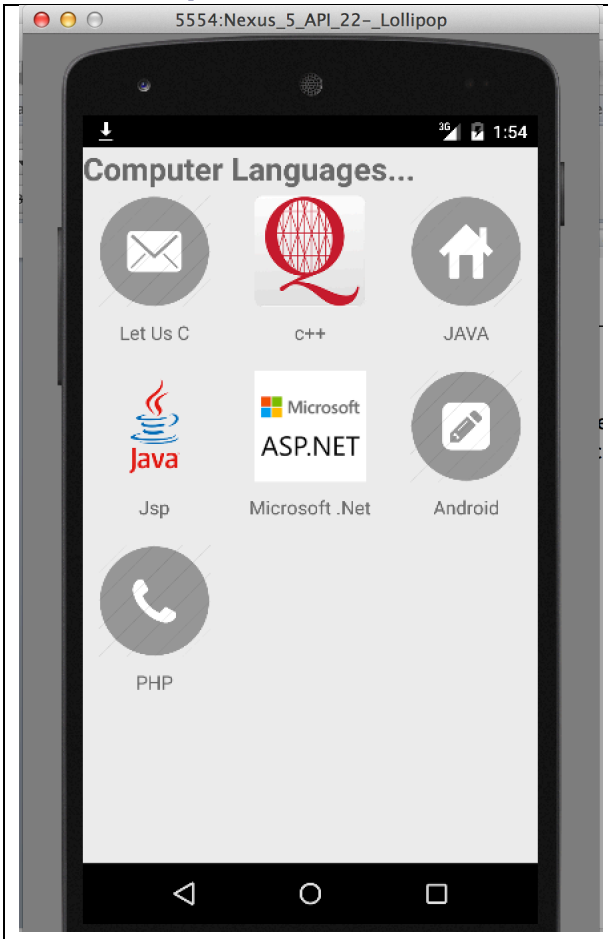

FIGURE 2 - FINAL OUTPUT

FIGURE 3 - FINAL OUTPUT 2

Step 1 Create Project: GridView

Open Android Studio and create a new project called "GridView". Refer to the 'Creating your first project' tutorial to help you create a project. Once created your project should look like figure 1. Switch from Design view to Text view (figure 4).

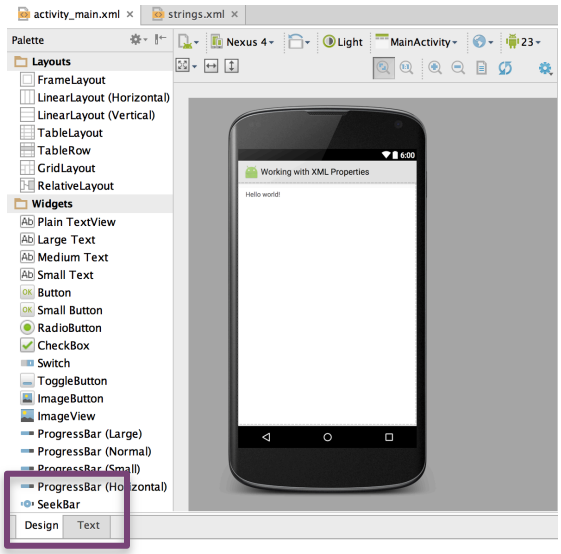

FIGURE 4 - OPEN PROJECT

Step 2 Find some programming languages images

Find some photos you'd like to use and save the image files into the project's res/drawable/ directory (figure 5).

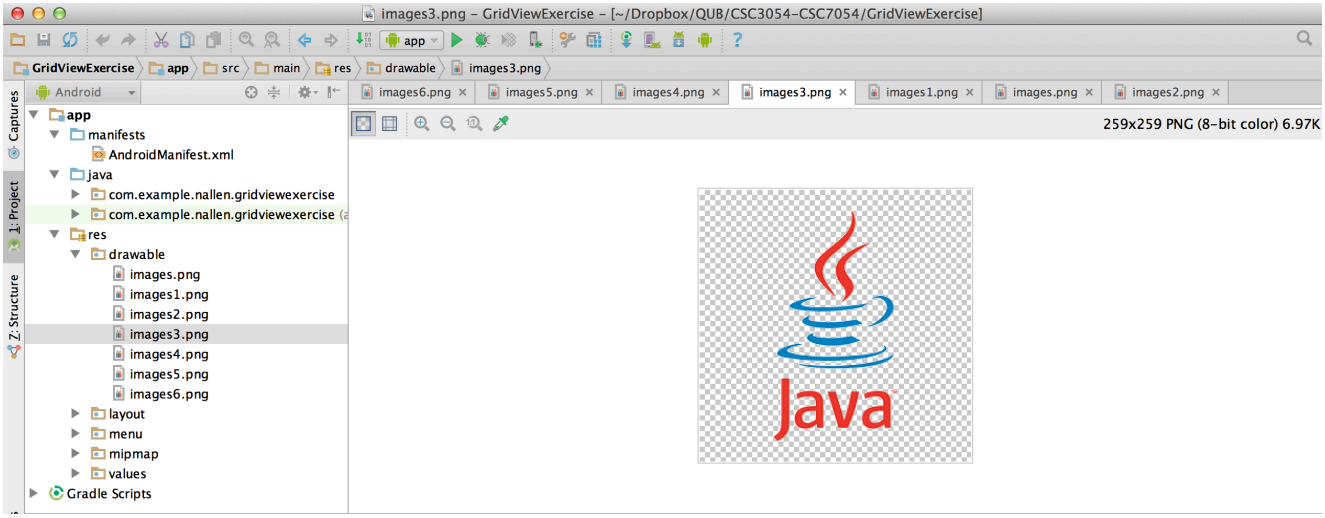

FIGURE 5 - INSERTING IMAGES INTO DRAWABLE FOLDER

Step 3 Update `activity_main.xml`

By default `activity_main.xml` is the default layout for the project. This layout will display the list of computer languages along with the associated image and name. Open the `res/layout/activity_main.xml` file and insert the following:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <GridView
        android:id="@+id/gridView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:verticalSpacing="10dp"
        android:horizontalSpacing="10dp"
        android:stretchMode="columnWidth"
        android:gravity="center"
        android:numColumns="3" >
    </GridView>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="@dimen/textSize"
        android:textStyle="bold"
        android:text="@string/title" />

</RelativeLayout>
```

Step 4: Update `Strings.xml`

```
<resources>
    <string name="app_name">GridView</string>
    <string name="action_settings">Settings</string>
    <string name="title">Computer Languages...</string>
</resources>
```

Step 5: Update `dimens.xml`

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="textSize">25sp</dimen>
</resources>
```

Step 6 Create another Layout resource file

Create another layout called "`programlist.xml`". This layout will have a `TextView` to display the computer language name and an `ImageView` to display the images associated with each computer language.

```
<?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" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_gravity="center"
        android:layout_width="88dp"
        android:layout_height="88dp"
        android:layout_marginTop="5dp"
        android:layout_marginBottom="5dp"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/textView1"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="15dp"
        android:layout_marginTop="5dp"
        android:layout_marginBottom="5dp"
        android:text="TextView" />
</LinearLayout>
```

Exercise 2 Add the functionality

Step 1 Update the `MainActivity.java` file

Declare a `GridView` reference, a `String` array with a number of programming languages and an `int` array with a number of references to the images that were stored in the `res/drawable` folder. Update the `onCreate` method as shown below. Make sure and inflate the layout with `activity_main` and initialize the `GridView` reference, linking it to the `GridView View Component` in the XML file.

```
import android.os.Bundle;
import android.app.Activity;
import android.widget.GridView;

public class MainActivity extends Activity {

    GridView gv;

    public static String[] prgmNameList =
            {"Let Us C", "c++", "JAVA", "Jsp", "Microsoft .Net", "Android", "PHP"};

    // references to images
    public static int[] prgmImages2 =
            {R.drawable.images, R.drawable.images1, R.drawable.images2,
             R.drawable.images3, R.drawable.images4, R.drawable.images5,
             R.drawable.images6};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        gv = (GridView) findViewById(R.id.gridView1);
        gv.setAdapter(new CustomAdapter(this, prgmNameList, prgmImages2));
    }
}
```

An `adapter` is attached to the `GridView` by use of the `setAdapter()` method. This method sets a custom adapter (`CustomAdapter`) as the source for all items to be displayed in the grid. The `CustomAdapter` is created in the next step.

Step 2: Create `CustomAdapter.java`

Create a new class called `CustomAdapter` that extends `BaseAdapter`.

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class CustomAdapter extends BaseAdapter {

    String[] result;
    Context context;
    int[] imageId;

    private static LayoutInflater inflater = null;

    public CustomAdapter(MainActivity mainActivity, String[] prgmNameList, int[]
        prgmImages) {
        result = prgmNameList;
        context = mainActivity;
        imageId = prgmImages;

        inflater = (LayoutInflater) context.
                getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
    @Override
    public int getCount() {
        return result.length;
    }

    @Override
    public Object getItem(int position) {
        return position;
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    public class Holder {
        TextView tv;
        ImageView img;
    }
```

```
    @Override
    // create a new ImageView for each item referenced by the Adapter
    public View getView(final int position, View convertView, ViewGroup parent) {
        Holder holder = new Holder();
        View rowView;

        rowView = inflater.inflate(R.layout.programlist, null);
        holder.tv = (TextView) rowView.findViewById(R.id.textView1);
        holder.img = (ImageView) rowView.findViewById(R.id.imageView1);

        holder.tv.setText(result[position]);
        holder.img.setImageResource(imageId[position]);

        rowView.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context, "You Clicked " + result[position],
                Toast.LENGTH_LONG).show();
            }
        });

        return rowView;
    }
}
```

To do something when an item in the grid is clicked, the `setOnItemClickListener()` method is passed a new `AdapterView.OnItemClickListener`.

This anonymous instance defines the `onItemClick()` callback method to show a `Toast` that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

This class implements some required methods inherited from `BaseAdapter`.

Method	Description	
`Public CustomAdapter (MainActivity mainActivity, String[] prgmNameList, int[] prgmImages)`	Constructor	Self-explanatory
`LayoutInflater inflater = (LayoutInflater)context.getSystemService (Context.LAYOUT_INFLATER_SERVICE);`	Layout Inflator	Instantiates a layout XML file into its corresponding View objects. It is never used directly. Instead, use `getLayoutInflater()` or `getSystemService(Class)` to retrieve a standard `LayoutInflater` instance that is already hooked up to the current context and correctly configured for the device you are running on.
`getCount()`	Required Method	Self-explanatory
`getItem(int)`	Required Method	Normally this method should return the actual object at the specified position in the adapter, but it's ignored for this example.
`getItemId(int)`	Required Method	This method should return the row id of the item, but it's not needed here.
`public class Holder { TextView tv; ImageView img; }`	Class	This is an inner class that defines layout of the `TextView` and `ImageView` for each image that is placed in the grid.
`getView().`	Required Method	This method creates a new `View` for each image added to the `CustomAdapter`. When this is called, a `View` is passed in, which is normally a recycled object (at least after this has been called once). The local `ImageView` and `TextView` is initialized with the recycled `View` object. At the end of the `getView()` method, the `position` integer passed into the method is used to select an image from the `imageId` array, which is set as the image resource for the `ImageView`.

Step 3 Check the `Mainfest.xml`

Navigate to the Mainfest.xml file and check that it reads as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nallen.gridviewexercise" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Step 4 Test and Run

When you run this application, you will be provide with a screen that will consist of a number of images displayed in a grid, that when clicked display a Toast message to the user.

Exercise 3 Extending the functionality

Let's extend the functionality of above example where we will show selected grid image in full screen. To achieve this we need to introduce a new activity. Just keep in mind for any activity we need perform all the steps like we have to implement an activity class, define that activity in `AndroidManifest.xml` file, define related layout and finally link that sub-activity with the main activity by it in the main activity class. So let's follow the steps to modify above example –

Step	Description
1	Create a new Activity class as `SingleViewActivity.java`
3	Create new layout file for the new activity under `res/layout/` folder. Let's name this XML file as `fullscreen.xml`.
4	Define your new activity in `AndroidManifest.xml` file using `<activity.../>` tag. An application can have one or more activities without any restrictions.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

Step 1 Create the new layout file `fullscreen.xml`

```
<?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" >

<ImageView android:id="@+id/SingleView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
</LinearLayout>
```

Step 2 Modify the content of `CustomAdapter.java`

This file can include each of the fundamental life cycle methods.

```
import android.content.Context;
import android.content.Intent;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class CustomAdapter extends BaseAdapter {
```

```
    String[] result;
    Context context;
    int[] imageId;

    private static LayoutInflater inflater = null;

    public CustomAdapter(MainActivity mainActivity, String[] prgmNameList, int[]
        prgmImages) {
        // TODO Auto-generated constructor stub
        result = prgmNameList;
        context = mainActivity;
        imageId = prgmImages;
        inflater = (LayoutInflater) context.
                getSystemService(Context.LAYOUT_INFLATER_SERVICE);

    }

    @Override
    public int getCount() {
        return result.length;
    }

    @Override
    public Object getItem(int position) {
        return position;
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    public class Holder {
        TextView tv;
        ImageView img;
    }

    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {
        Holder holder = new Holder();
        View rowView;

        rowView = inflater.inflate(R.layout.programlist, null);
        holder.tv = (TextView) rowView.findViewById(R.id.textView1);
        holder.img = (ImageView) rowView.findViewById(R.id.imageView1);

        holder.tv.setText(result[position]);
        holder.img.setImageResource(imageId[position]);

        rowView.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context, "You Clicked " + result[position],
                Toast.LENGTH_LONG).show();
                final int pos = position;
                goToNextScreen(context, pos);
            }
```

```
        });
        return rowView;
    }

    public void goToNextScreen(Context context, int position) {
        Intent intent = new Intent(context, SingleViewActivity.class);
        intent.putExtra("position", position);
        context.startActivity(intent);
    }
}
```

Step 3 Create a new activity `SingleViewActivity.java`

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;

public class SingleViewActivity extends Activity {

    ImageView iv;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.fullscreen);

        // Get intent data
        Intent i = getIntent();

        // Selected image id
        int position = i.getExtras().getInt("position");

        setInfo(position);
    }

    public void setInfo(int position)
    {

        iv = (ImageView) findViewById(R.id.SingleView);
        iv.setImageResource(MainActivity.prgmImages2[position]);

    }
}
```

Step 4 Update the `AndroidManifest.xml` with the new `Activity`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nallen.gridviewexercise" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".SingleViewActivity"></activity>

    </application>

</manifest>
```

Step 5 Test and Run

When you run this application, you will be provide with a screen that will consist of a number of images displayed in a grid, that when clicked display a Toast message to the user.

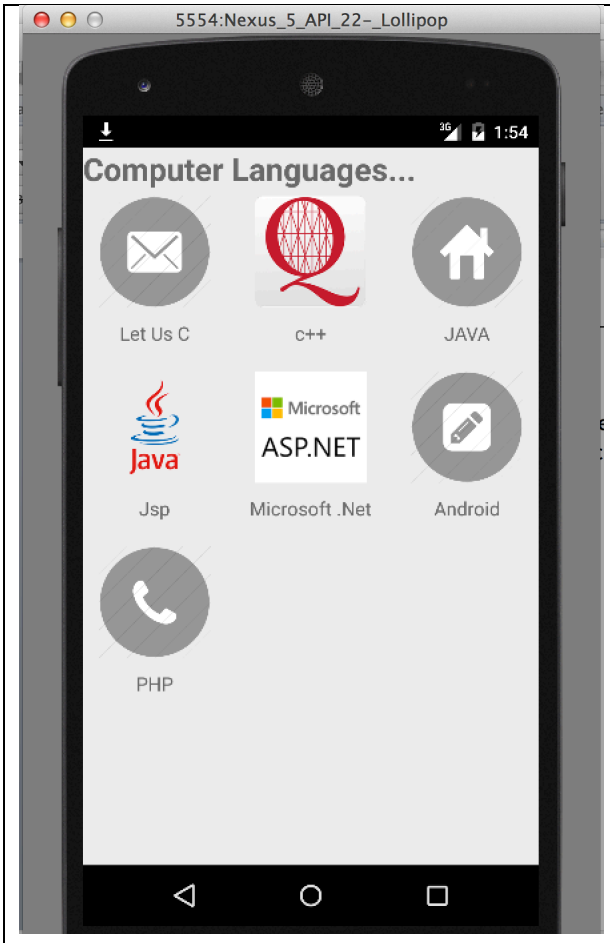

FIGURE 6 SCREEN 1

FIGURE 7 SCREEN 2