

## 38. An Android RecyclerView and CardView Tutorial

In this chapter an example project will be created that makes use of both the CardView and RecyclerView components to create a scrollable list of cards. The completed app will display a list of cards containing images and text. In addition to displaying the list of cards, the project will be implemented such that selecting a card causes a messages to be displayed to the user indicating which card was tapped.

### 38.1 Creating the CardDemo Project

Create a new project in Android Studio, entering *CardDemo* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich) and continue to proceed through the screens.

In the next chapter, the scroll handling features of the AppBar, Toolbar and CoordinatorLayout layout will be demonstrated using this project. On the activity selection screen, therefore, request the creation of a Basic Activity named *CardDemoActivity* with a corresponding layout file named *activity\_card\_demo*. Click on the *Finish* button to initiate the project creation process.

Once the project has been created, load the *content\_card\_demo.xml* file into the Layout Editor tool and select and delete the “Hello World” TextView object.

### 38.2 Removing the Floating Action Button

Since the Basic Activity was selected, the layout includes a floating action button which is not required for this project. Load the *activity\_card\_demo.xml* layout file into the Layout Editor tool, select the floating action button and tap the keyboard delete key to remove the object from the layout. Edit the *CardDemoActivity.java* file and remove the floating action button code from the onCreate method as follows:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_card_demo);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab =
    (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view, "Replace with your own action",
            Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        }
    });
}

```

## 38.3 Adding the RecyclerView and CardView Libraries

Within the Project tool window locate and select the module level *build.gradle* file and modify the dependencies section of the file to add the support library dependencies for the RecyclerView and CardView:

```

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-
layout:1.0.2'
    implementation 'com.android.support:design:26.1.0'
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    implementation 'com.android.support:cardview-v7:26.1.0'
    .
    .
}

```

When prompted to do so, resync the new Gradle build configuration by clicking on the *Sync Now* link in the warning bar.

## 38.4 Designing the CardView Layout

The layout of the views contained within the cards will be defined within a separate XML layout file. Within the Project tool window right click on the *app* -> *res* -> *layout* entry and select the *New* -> *Layout resource file* menu

option. In the New Resource Dialog enter *card\_layout* into the *File name:* field and *android.support.v7.widget.CardView* into the root element field before clicking on the OK button.

Load the *card\_layout.xml* file into the Layout Editor tool, switch to Text mode and modify the layout so that it reads as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/card_view"
    android:layout_margin="5dp"
    card_view:cardBackgroundColor="#81C784"
    card_view:cardCornerRadius="12dp"
    card_view:cardElevation="3dp"
    card_view:contentPadding="4dp" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp" >

        <ImageView
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:id="@+id/item_image"
            android:layout_alignParentStart="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginEnd="16dp"
            android:layout_marginRight="16dp" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/item_title"
            android:layout_toEndOf="@+id/item_image"
            android:layout_toRightOf="@+id/item_image"
            android:layout_alignParentTop="true"
            android:textSize="30sp" />
```

```

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/item_detail"
            android:layout_toEndOf="@+id/item_image"
            android:layout_toRightOf="@+id/item_image"
            android:layout_below="@+id/item_title" />

    </RelativeLayout>
</android.support.v7.widget.CardView>

```

## 38.5 Adding the RecyclerView

Select the *activity\_card\_demo.xml* layout file and modify it to add the RecyclerView component immediately before the AppBarLayout:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.ebookfrenzy.ccarddemo.CardDemoActivity">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />

    <android.support.design.widget.AppBarLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:theme="@style/AppTheme.AppBarOverlay">

```

.  
.  
.

## 38.6 Creating the RecyclerView Adapter

As outlined in the previous chapter, the RecyclerView needs to have an adapter to handle the creation of the list items. Add this new class to the

project by right-clicking on the *app* -> *java* -> *com.ebookfrenzy.ccarddemo* entry in the Project tool window and selecting the *New* -> *Java Class* menu option. In the Create New Class dialog, enter *RecyclerViewAdapter* into the *Name:* field before clicking on the OK button to create the new Java class file.

Edit the new *RecyclerViewAdapter.java* file to add some import directives and to declare that the class now extends *RecyclerView.Adapter*. Rather than create a separate class to provide the data to be displayed, some basic arrays will also be added to the adapter to act as the data for the app:

```
package com.ebookfrenzy.ccarddemo;

import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private String[] titles = {"Chapter One",
        "Chapter Two",
        "Chapter Three",
        "Chapter Four",
        "Chapter Five",
        "Chapter Six",
        "Chapter Seven",
        "Chapter Eight"};

    private String[] details = {"Item one details",
        "Item two details", "Item three details",
        "Item four details", "Item five details",
        "Item six details", "Item seven details",
        "Item eight details"};

    private int[] images = { R.drawable.android_image_1,
        R.drawable.android_image_2,
        R.drawable.android_image_3,
        R.drawable.android_image_4,
        R.drawable.android_image_5,
        R.drawable.android_image_6,
```

```

        R.drawable.android_image_7,
        R.drawable.android_image_8 };
    }

```

Within the `RecyclerViewAdapter` class we now need our own implementation of the `ViewHolder` class configured to reference the view elements in the *card\_layout.xml* file. Remaining within the *RecyclerViewAdapter.java* file implement this class as follows:

```

.
.
public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerView.ViewHolder> {
.
.
    class ViewHolder extends RecyclerView.ViewHolder {

        public ImageView itemImage;
        public TextView itemTitle;
        public TextView itemDetail;

        public ViewHolder(View itemView) {
            super(itemView);
            itemImage =
                (ImageView)itemView.findViewById(R.id.item_image);
            itemTitle =
                (TextView)itemView.findViewById(R.id.item_title);
            itemDetail =
                (TextView)itemView.findViewById(R.id.item_detail);
        }
    }
.
.
}

```

The `ViewHolder` class contains an `ImageView` and two `TextView` variables together with a constructor method that initializes those variables with references to the three view items in the *card\_layout.xml* file.

The next item to be added to the *RecyclerViewAdapter.java* file is the implementation of the *onCreateViewHolder()* method:

```

@Override
public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
    View v = LayoutInflater.from(viewGroup.getContext())

```

```

        .inflate(R.layout.card_layout, viewGroup, false);
ViewHolder viewHolder = new ViewHolder(v);
return viewHolder;
}

```

This method will be called by the RecyclerView to obtain a ViewHolder object. It inflates the view hierarchy *card\_layout.xml* file and creates an instance of our ViewHolder class initialized with the view hierarchy before returning it to the RecyclerView.

The purpose of the *onBindViewHolder()* method is to populate the view hierarchy within the ViewHolder object with the data to be displayed. It is passed the ViewHolder object and an integer value indicating the list item that is to be displayed. This method should now be added, using the item number as an index into the data arrays. This data is then displayed on the layout views using the references created in the constructor method of the ViewHolder class:

```

@Override
public void onBindViewHolder(ViewHolder viewHolder, int i) {
    viewHolder.itemTitle.setText(titles[i]);
    viewHolder.itemDetail.setText(details[i]);
    viewHolder.itemImage.setImageResource(images[i]);
}

```

The final requirement for the adapter class is an implementation of the *getItem()* method which, in this case, simply returns the number of items in the *titles* array:

```

@Override
public int getItemCount() {
    return titles.length;
}

```

## 38.7 Adding the Image Files

In addition to the two TextViews, the card layout also contains an ImageView on which the RecyclerView adapter has been configured to display images. Before the project can be tested these images must be added. The images that will be used for the project are named *android\_image\_<n>.jpg* and can be found in the *project\_icons* folder of the sample code download available from the following URL:

<http://www.ebookfrenzy.com/retail/androidstudio30/index.php>

Locate these images in the file system navigator for your operating system and select and copy the eight images. Right click on the *app -> res -> drawable* entry in the Project tool window and select Paste to add the files to the folder:

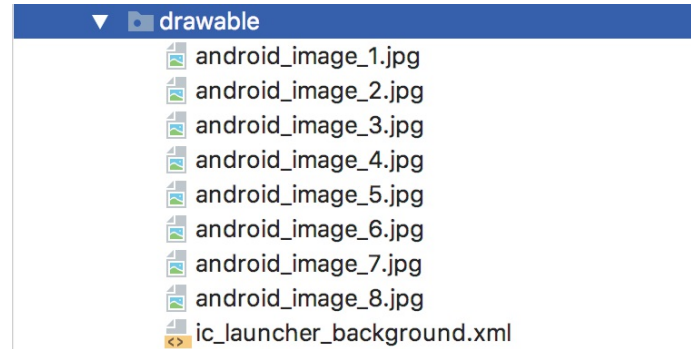


Figure 38-1

## 38.8 Initializing the RecyclerView Component

At this point the project consists of a RecyclerView instance, an XML layout file for the CardView instances and an adapter for the RecyclerView. The last step before testing the progress so far is to initialize the RecyclerView with a layout manager, create an instance of the adapter and assign that instance to the RecyclerView object. For the purposes of this example, the RecyclerView will be configured to use the LinearLayoutManager layout option. Edit the *CardDemoActivity.java* file and modify the *onCreate()* method to implement this initialization code:

```
package com.ebookfrenzy.ccarddemo;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;

public class CardDemoActivity extends AppCompatActivity {

    RecyclerView recyclerView;
    RecyclerView.LayoutManager layoutManager;
    RecyclerView.Adapter adapter;
```



```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_card_demo);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    recyclerView =
        (RecyclerView) findViewById(R.id.recycler_view);

    layoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(layoutManager);

    adapter = new RecyclerViewAdapter();
    recyclerView.setAdapter(adapter);
}
.
.
}

```

## 38.9 Testing the Application

Compile and run the app on a physical device or emulator session and scroll through the different card items in the list:

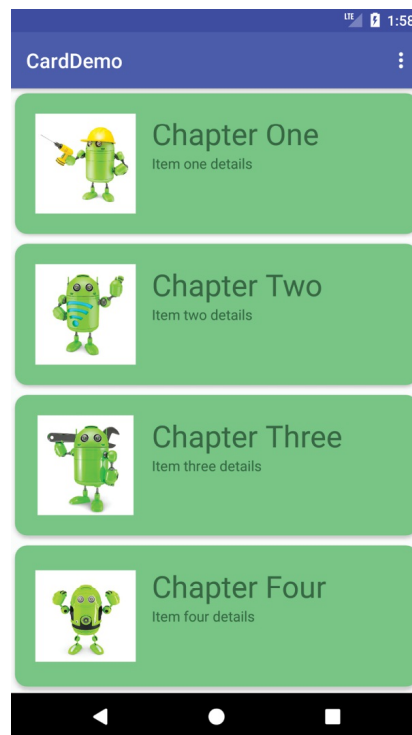


Figure 38-2

## 38.10 Responding to Card Selections

The last phase of this project is to make the cards in the list selectable so that clicking on a card triggers an event within the app. For this example, the cards will be configured to present a message on the display when tapped by the user. To respond to clicks, the ViewHolder class needs to be modified to assign an `onClick` listener on each item view. Edit the *RecyclerViewAdapter.java* file and modify the ViewHolder class declaration so that it reads as follows:

```
import android.support.design.widget.Snackbar;
.
.
.
class ViewHolder extends RecyclerView.ViewHolder{

    public ImageView itemImage;
    public TextView itemTitle;
    public TextView itemDetail;

    public ViewHolder(View itemView) {
        super(itemView);
        itemImage =
        (ImageView) itemView.findViewById(R.id.item_image);
        itemTitle = (TextView) itemView.findViewById(R.id.item_title);
        itemDetail =
        (TextView) itemView.findViewById(R.id.item_detail);

        itemView.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {

                }
        });
    }
}
```

Within the body of the `onClick` handler, code can now be added to display a message indicating that the card has been clicked. Given that the actions performed as a result of a click will likely depend on which card was tapped it is also important to identify the selected card. This information can be obtained via a call to the `getAdapterPosition()` method of the *RecyclerView.ViewHolder* class. Remaining within the *RecyclerViewAdapter.java* file, add code to the *onClick* handler so it reads as follows:

```

@Override
public void onClick(View v) {

    int position = getAdapterPosition();

    Snackbar.make(v, "Click detected on item " + position,
        Snackbar.LENGTH_LONG)
        .setAction("Action", null).show();
    }
});

```

The last task is to enable the material design ripple effect that appears when items are tapped within Android applications. This simply involves the addition of some properties to the declaration of the `CardView` instance in the *card\_layout.xml* file as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/card_view"
    android:layout_margin="5dp"
    card_view:cardBackgroundColor="#81C784"
    card_view:cardCornerRadius="12dp"
    card_view:cardElevation="3dp"
    card_view:contentPadding="4dp"
    android:foreground="?selectableItemBackground"
    android:clickable="true" >

```

Run the app once again and verify that tapping a card in the list triggers both the standard ripple effect at the point of contact and the appearance of a `Snackbar` reporting the number of the selected item.

## 38.1 Summary

This chapter has worked through the steps involved in combining the `CardView` and `RecyclerView` components to display a scrollable list of card based items. The example also covered the detection of clicks on list items, including the identification of the selected item and the enabling of the ripple effect visual feedback on the tapped `CardView` instance.

# 39. Working with the AppBar and Collapsing Toolbar Layouts

In this chapter we will be exploring the ways in which the app bar within an activity layout can be customized and made to react to the scrolling events taking place within other views on the screen. By making use of the CoordinatorLayout in conjunction with the AppBarLayout and CollapsingToolbarLayout containers, the app bar can be configured to display an image and to animate in and out of view. An upward scrolling motion on a list, for example, can be configured so that the app bar recedes from view and then reappears when a downward scrolling motion is performed.

Beginning with an overview of the elements that can comprise an app bar, this chapter will then work through a variety of examples of app bar configuration.

## 39.1 The Anatomy of an AppBar

The app bar is the area that appears at the top of the display when an app is running and can be configured to contain a variety of different items including the status bar, toolbar, tab bar and a flexible space area. [Figure 39-1](#), for example, shows an app bar containing a status bar, toolbar and tab bar:

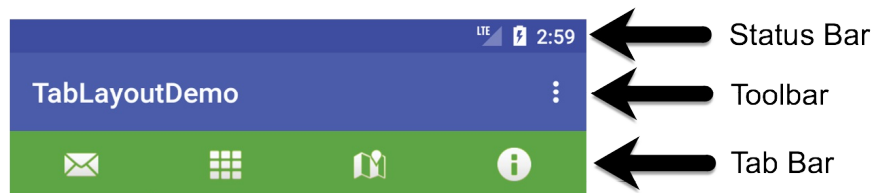


Figure 39-1

The flexible space area can be filled by a blank background color, or as shown in [Figure 39-2](#), an image displayed on an ImageView object:

