CSCI 322/522 **15. Android RecyclerView and CardView**

Working with the RecyclerView and CardView Widgets

- The RecyclerView and CardView widgets work together to provide scrollable lists of information to the user in which the information is presented in the form of individual cards.
- Details of both classes will be covered in this chapter before working through the design and implementation of an example project.

An Overview of the RecyclerView

- The purpose of the RecyclerView is to allow information to be presented to the user in the form of a scrollable list. (It is a lot like the ListView to be described in an upcoming chapter.)
- The RecyclerView, however, provides a number of advantages over the ListView.
- In particular, the RecyclerView is significantly more efficient in the way it manages the views that make up a list, essentially reusing existing views that make up list items as they scroll off the screen instead if creating new ones (hence the name "recycler").
- This both increases the performance and reduces the resources used by a list, a feature that is of particular benefit when presenting large amounts of data to the user.
- The RecyclerView also provides a choice of three built-in layout managers to control the way in which the list items are presented to the user:
  - LinearLayoutManager – The list items are presented as either a horizontal or vertical scrolling list.

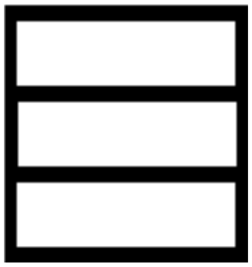


Figure 15-1

  - GridLayoutManager – The list items are presented in grid format. This manager is best used when the list items are of uniform size.

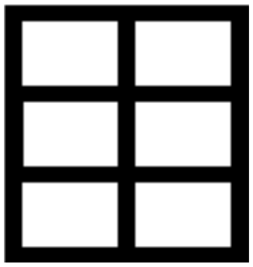


Figure 15-2

  - StaggeredGridLayoutManager - The list items are presented in a staggered grid format.

This manager is best used when the list items are not of uniform size.

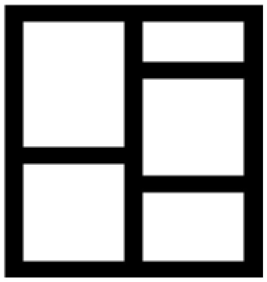


Figure 15-3

- For situations where none of the three built-in managers provide the necessary layout, custom layout managers may be implemented by subclassing the RecyclerView.LayoutManager class.

- Each list item displayed in a RecyclerView is created as an instance of the ViewHolder class.

- The ViewHolder instance contains everything necessary for the RecyclerView to display the list item, including the information to be displayed and the view layout used to display the item.

- As with the ListView, the RecyclerView depends on an adapter to act as the intermediary between the RecyclerView instance and the data that is to be displayed to the user.

- The adapter is created as a subclass of the RecyclerView.Adapter class and must, at a minimum, implement the following methods, which will be called at various points by the RecyclerView object to which the adapter is assigned:

  - getItemCount() – This method must return a count of the number of items that are to be displayed in the list.

  - onCreateViewHolder() – This method creates and returns a ViewHolder object initialized with the view that is to be used to display the data.

    This view is typically created by inflating the XML layout file.

  - onBindViewHolder() – This method is passed the ViewHolder object created by the onCreateViewHolder() method together with an integer value indicating the list item that is about to be displayed.

- Contained within the ViewHolder object is the layout assigned by the onCreateViewHolder() method.

- It is the responsibility of the onBindViewHolder() method to populate the views in the layout with the text and graphics corresponding to the specified item and to return the object to the RecyclerView where it will be presented to the user.

- Adding a RecyclerView to a layout is simply a matter of adding the appropriate element to the XML content layout file of the activity in which it is to appear.

(continued)

- For example:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  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"
  app:layout_behavior="@string/appbar_scrolling_view_behavior"
  tools:context=".MainActivity"
  tools:showIn="@layout/activity_card_demo">

<androidx.recyclerview.widget.RecyclerView
  android:id="@+id/recycler_view"
  android:layout_width="0dp" android:layout_height="0dp"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  tools:listItem="@layout/card_layout" />

</androidx.constraintlayout.widget.ConstraintLayout>
.
.
```

- In the above example the RecyclerView has been embedded into the CoordinatorLayout of a main activity layout file along with the AppBar and Toolbar.

- This provides some additional features, such as configuring the Toolbar and AppBar to scroll off the screen when the user scrolls up within the RecyclerView.

An Overview of the CardView

- The CardView class is a user interface view that allows information to be presented in groups using a card metaphor.

- Cards are usually presented in lists using a RecyclerView instance and may be configured to appear with shadow effects and rounded corners.

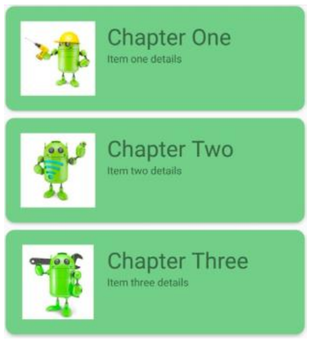


Figure 15-4

- Figure 15-4 above, for example, shows three CardView instances configured to display a layout consisting of an ImageView and two TextViews.

- The user interface layout to be presented with a CardView instance is defined within an XML layout resource file and loaded into the CardView at runtime.

- The CardView layout can contain a layout of any complexity using the standard layout managers such as RelativeLayout and LinearLayout.

- The following XML layout file represents a card view layout consisting of a RelativeLayout and a single ImageView.

- The card is configured to be elevated to create shadowing effect and to appear with rounded corners:

```
<?xml version="1.0" encoding="utf-8"?>
  <androidx.cardview.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    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_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginRight="16dp" />

  </RelativeLayout>
</androidx.cardview.widget.CardView>
```

- When combined with the RecyclerView to create a scrollable list of cards, the onCreateViewHolder() method of the recycler view inflates the layout resource file for the card, assigns it to the ViewHolder instance and returns it to the RecyclerView instance.

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 messages to be displayed to the user indicating which card was tapped.

Creating the CardDemo Project

- Select the Start a new Android Studio project quick start option from the welcome screen and, within the resulting new project dialog, choose the Basic Activity template before clicking on the Next button.

- Enter CardDemo into the Name field and specify edu.niu.your_Z-ID.carddemo as the package name.

- Before clicking on the Finish button, change the Minimum API level setting to API 26: Android 8.0 (Oreo) and the Language menu to Java.

- Once the project has been created, load the content_main.xml file into the Layout Editor tool and select and delete the nav_host_fragment object.

Modifying the Basic Activity Project

- Since the Basic Activity was selected, the layout includes a floating action button which is not required for this project.

- Load the activity_main.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 MainActivity.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 = findViewById(R.id.toolbar);
  setSupportActionBar(toolbar);

  FloatingActionButton fab = 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();
    }
  });
}
```

- The project will also not require the default navigation features, so open the content_main.xml file and delete the NavHostFragment object from the layout so that only the ConstraintLayout parent remains.

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 androidx.cardview.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 Code mode and modify the layout so that it reads as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="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"
  app:cardBackgroundColor="#81C784"
  app:cardCornerRadius="12dp"
  app:cardElevation="3dp"
  app:contentPadding="4dp" >

  <androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/relativeLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">

    <ImageView
      android:id="@+id/item_image"
      android:layout_width="100dp"
      android:layout_height="100dp"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />

    <TextView
      android:id="@+id/item_title"
      android:layout_width="236dp"
      android:layout_height="39dp"
      android:layout_marginStart="16dp"
      android:textSize="30sp"
      app:layout_constraintLeft_toRightOf="@+id/item_image"
```

```
        app:layout_constraintStart_toEndOf="@+id/item_image"
        app:layout_constraintTop_toTopOf="parent" />

      <TextView
        android:id="@+id/item_detail"
        android:layout_width="236dp"
        android:layout_height="16dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        app:layout_constraintLeft_toRightOf="@+id/item_image"
        app:layout_constraintStart_toEndOf="@+id/item_image"
        app:layout_constraintTop_toBottomOf="@+id/item_title" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>
```

Adding the RecyclerView

- Select the content_main.xml layout file and drag and drop a RecyclerView object from the Containers section of the palette onto the layout so that it is positioned in the center of the screen where it should automatically resize to fill the entire screen.

- Use the Infer constraints toolbar button to add any missing layout constraints to the view.

- Using the Attributes tool window, change the ID of the RecyclerView instance to recyclerView and the layout_width and layout_height properties to match_constraint.

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 > edu.niu.your_Z-ID.carddemo entry in the Project tool window and selecting the New > Java Class menu option.

- In the new class dialog, enter RecyclerAdapter into the Name field and select Class from the list before tapping the Return keyboard key to create the new Java class file.

- Edit the new RecyclerAdapter.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 edu.niu.your_Z-ID.carddemo;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.recyclerview.widget.RecyclerView;

public class RecyclerAdapter
  extends RecyclerView.Adapter<RecyclerAdapter. 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 RecyclerAdapter 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 RecyclerAdapter.java file implement this class as follows:

```
public class RecyclerAdapter
  extends RecyclerView.Adapter<RecyclerAdapter. ViewHolder>
{
  .
  .
  static class ViewHolder extends RecyclerView.ViewHolder
  {
    ImageView itemImage;
    TextView itemTitle;
    TextView itemDetail;

    ViewHolder(View itemView)
    {
      super(itemView);
      itemImage = itemView.findViewById(R.id.item_image);
      itemTitle = itemView.findViewById(R.id.item_title);
      itemDetail = 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 RecyclerAdapter.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;
}
```

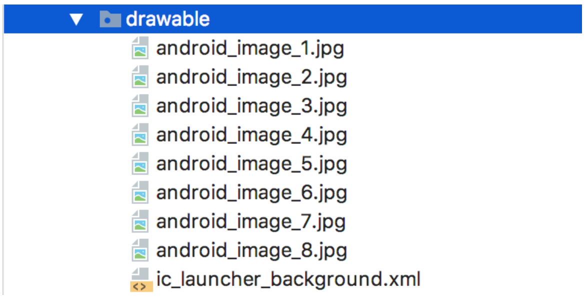
(continued)

Adding the Image Files

- In addition to the two TextViews, the card layout also contains an ImageView on which the Recycler 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:

  `https://www.ebookfrenzy.com/retail/androidstudio40/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:



```
▼ drawable
    android_image_1.jpg
    android_image_2.jpg
    android_image_3.jpg
    android_image_4.jpg
    android_image_5.jpg
    android_image_6.jpg
    android_image_7.jpg
    android_image_8.jpg
    ic_launcher_background.xml
```

Figure 15-5

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 MainActivity.java file and modify the onCreate() method to implement this initialization code:

```
package edu.niu.your_Z-ID.carddemo;
.
.
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity 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 = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    recyclerView = findViewById(R.id.recyclerView);

    layoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(layoutManager);
    adapter = new RecyclerAdapter();
    recyclerView.setAdapter(adapter);
  }
  .
  .
}
```

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:

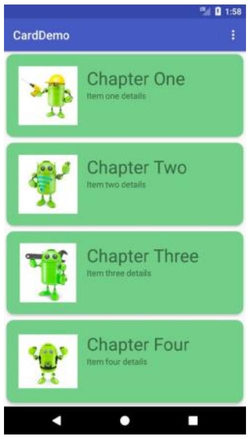


Figure 15-6

(continued)

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 onClickListener on each item view.

- Edit the RecyclerAdapter.java file and modify the ViewHolder class declaration so that it reads as follows:

```
.
.
import com.google.android.material.snackbar.Snackbar;
.
.
class ViewHolder extends RecyclerView.ViewHolder
{

  ImageView itemImage;
  TextView itemTitle;
  TextView itemDetail;

  ViewHolder(View itemView)
  {
    super(itemView);
    itemImage = itemView.findViewById(R.id.item_image);
    itemTitle = itemView.findViewById(R.id.item_title);
    itemDetail = 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 RecyclerAdapter.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 chapter " + (position+1),
              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"?>
<androidx.cardview.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"
  app:cardBackgroundColor="#81C784"
  app:cardCornerRadius="12dp"
  app:cardElevation="3dp"
  app: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.

Summary

- This chapter has introduced the Android RecyclerView and CardView components.

- The RecyclerView provides a resource efficient way to display scrollable lists of views within an Android app.

- The CardView is useful when presenting groups of data (such as a list of names and addresses) in the form of cards.

- As previously outlined, and demonstrated in the tutorial contained in the next chapter, the RecyclerView and CardView are particularly useful when combined.

- 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.