CSCI 322/522 12. Android Fragments

An Introduction to Android Fragments

- As you progress through the chapters of this book it will become increasingly evident that many of the design concepts behind the Android system were conceived with the goal of promoting reuse of, and interaction between, the different elements that make up an application.

- One such area that will be explored in this chapter involves the use of Fragments.

- This chapter will provide an overview of the basics of fragments in terms of what they are and how they can be created and used within applications.

- The next chapter will work through a tutorial designed to show fragments in action when developing applications in Android Studio, including the implementation of communication between fragments.

What is a Fragment?

- A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

- Fragments can be assembled to create an activity during the application design phase, and added to or removed from an activity during application runtime to create a dynamically changing user interface.

- Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.

- That being said, however, a fragment can be thought of as a functional "sub-activity" with its own lifecycle similar to that of a full activity.

- Fragments are stored in the form of XML layout files and may be added to an activity either by placing appropriate <fragment> elements in the activity's layout file, or directly through code within the activity's class implementation.

Creating a Fragment

- The two components that make up a fragment are an XML layout file and a corresponding Java class.

- The XML layout file for a fragment takes the same format as a layout for any other activity layout and can contain any combination and complexity of layout managers and views.

- The following XML layout, for example, is for a fragment consisting simply of a RelativeLayout with a red background containing a single TextView:

```
<?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:background="@color/red" >
```

```
    <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="@string/fragone_label_text"
      android:textAppearance="?android:attr/textAppearanceLarge" />
  </RelativeLayout>
```

- The corresponding class to go with the layout must be a subclass of the Android Fragment class.

- This class should, at a minimum, override the onCreateView() method which is responsible for loading the fragment layout.

- For example:

```
package com.example.myfragmentdemo;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;

public class FragmentOne extends Fragment
{

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState)
  {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_one_layout, container, false);
  }
}
```

- In addition to the onCreateView() method, the class may also override the standard lifecycle methods.

- Once the fragment layout and class have been created, the fragment is ready to be used within application activities.

Adding a Fragment to an Activity Using the Layout XML File

- Fragments may be incorporated into an activity either by writing Java code or by embedding the fragment into the activity's XML layout file.

- Regardless of the approach used, a key point to be aware of is that when the support library is being used for compatibility with older Android releases, any activities using fragments must be implemented as a subclass of FragmentActivity instead of the AppCompatActivity class:

```
package com.example.myfragmentdemo;

import android.os.Bundle;
import androidx.fragment.app.FragmentActivity;
import android.view.Menu;

public class MainActivity extends FragmentActivity
{

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

- Fragments are embedded into activity layout files using the <fragment> element.

- The following example layout embeds the fragment created in the previous section of this chapter into an activity layout:

```
<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"
  tools:context=".MainActivity" >

  <fragment
    android:id="@+id/fragment_one"
    android:name="com.example.myfragmentdemo.myfragmentdemo.FragmentOne"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    tools:layout="@layout/fragment_one_layout" />

</RelativeLayout>
```

- The key properties within the <fragment> element are android:name, which must reference the class associated with the fragment, and tools:layout, which must reference the XML resource file containing the layout of the fragment.

- Once added to the layout of an activity, fragments may be viewed and manipulated within the Android Studio Layout Editor tool.

(continued)

- Figure 12-1, for example, shows the above layout with the embedded fragment within the Android Studio Layout Editor:

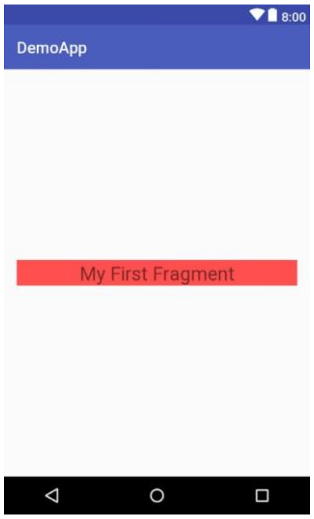


Figure 12-1

Adding and Managing Fragments in Code

- The ease of adding a fragment to an activity via the activity's XML layout file comes at the cost of the activity not being able to remove the fragment at runtime.

- In order to achieve full dynamic control of fragments during runtime, those activities must be added via code.

- This has the advantage that the fragments can be added, removed and even made to replace one another dynamically while the application is running.

- When using code to manage fragments, the fragment itself will still consist of an XML layout file and a corresponding class.

- The difference comes when working with the fragment within the hosting activity. There is a standard sequence of steps when adding a fragment to an activity using code:

    - Create an instance of the fragment's class.
    - Pass any additional intent arguments through to the class instance.
    - Obtain a reference to the fragment manager instance.
    - Call the beginTransaction() method on the fragment manager instance. This returns a fragment transaction instance.
    - Call the add() method of the fragment transaction instance, passing through as arguments the resource ID of the view that is to contain the fragment and the fragment class instance.
    - Call the commit() method of the fragment transaction.

- The following code, for example, adds a fragment defined by the FragmentOne class so that it appears in the container view with an ID of LinearLayout1:

```
FragmentOne firstFragment = new FragmentOne();
firstFragment.setArguments(getIntent().getExtras());
```

```
FragmentManager fragManager = getSupportFragmentManager();
FragmentTransaction transaction = fragManager.beginTransaction();

transaction.add(R.id.LinearLayout1, firstFragment);
transaction.commit();
```

- The above code breaks down each step into a separate statement for the purposes of clarity.

- The last four lines can, however, be abbreviated into a single line of code as follows:

```
getSupportFragmentManager().beginTransaction()
  .add(R.id.LinearLayout1, firstFragment).commit();
```

- Once added to a container, a fragment may subsequently be removed via a call to the remove() method of the fragment transaction instance, passing through a reference to the fragment instance that is to be removed:

```
transaction.remove(firstFragment);
```

- Similarly, one fragment may be replaced with another by a call to the replace() method of the fragment transaction instance.

- This takes as arguments the ID of the view containing the fragment and an instance of the new fragment.

- The replaced fragment may also be placed on what is referred to as the back stack so that it can be quickly restored in the event that the user navigates back to it.

- This is achieved by making a call to the addToBackStack() method of the fragment transaction object before making the commit() method call:

```
FragmentTwo secondFragment = new FragmentTwo();
transaction.replace(R.id.LinearLayout1, secondFragment);
transaction.addToBackStack(null);
transaction.commit();
```

Handling Fragment Events

- As previously discussed, a fragment is very much like a sub-activity with its own layout, class and lifecycle.

- The view components (such as buttons and text views) within a fragment are able to generate events just like those in a regular activity.

- This raises the question as to which class receives an event from a view in a fragment; the fragment itself, or the activity in which the fragment is embedded.

- The answer to this question depends on how the event handler is declared.

- In Chapter 11 – Android Event Handling, two approaches to event handling were discussed.

- The first method involved configuring an event listener and callback method within the code of the activity.

- For example:

```
button.setOnClickListener(
  new Button.OnClickListener()
  {
    public void onClick(View v)
    {
      // Code to be performed when
      // the button is clicked
    }
  }
);
```

- In the case of intercepting click events, the second approach involved setting the android:onClick property within the XML layout file:

```
<Button
  android:id="@+id/button1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:onClick="onClick"
  android:text="Click me" />
```

- The general rule for events generated by a view in a fragment is that if the event listener was declared in the fragment class using the event listener and callback method approach, then the event will be handled first by the fragment.

- If the android:onClick resource is used, however, the event will be passed directly to the activity containing the fragment.

Implementing Fragment Communication

- Once one or more fragments are embedded within an activity, the chances are good that some form of communication will need to take place both between the fragments and the activity, and between one fragment and another.

- In fact, good practice dictates that fragments do not communicate directly with one another.

- All communication should take place via the encapsulating activity.

- In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it.

- Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

- Communicating in the other direction (from fragment to activity) is a little more complicated.

- In the first instance, the fragment must define a listener interface, which is then implemented within the activity class.

- For example, the following code declares an interface named ToolbarListener on a fragment class named ToolbarFragment.

- The code also declares a variable in which a reference to the activity will later be stored:

```
public class ToolbarFragment extends Fragment
{
  ToolbarListener activityCallback;

  public interface ToolbarListener
  {
    public void onButtonClick(int position, String text);
  }
  .
  .
}
```

- The above code dictates that any class that implements the ToolbarListener interface must also implement a callback method named onButtonClick which, in turn, accepts an integer and a String as arguments.

- Next, the onAttach() method of the fragment class needs to be overridden and implemented.

- This method is called automatically by the Android system when the fragment has been initialized and associated with an activity.

- The method is passed a reference to the activity in which the fragment is contained. The method must store a local reference to this activity and verify that it implements the ToolbarListener interface:

```
@Override
public void onAttach(Context context)
{
  super.onAttach(context);

  try
  {
    activityCallback = (ToolbarListener) activity;
  }
  catch (ClassCastException e)
  {
    throw new ClassCastException(activity.toString()
      + " must implement ToolbarListener");
  }
}
```

- Upon execution of this example, a reference to the activity will be stored in the local activityCallback variable, and an exception will be thrown if that activity does not implement the ToolbarListener interface.

- The next step is to call the callback method of the activity from within the fragment.

- When and how this happens is entirely dependent on the circumstances under which the activity needs to be contacted by the fragment.

- The following code, for example, calls the callback method on the activity when a button is clicked:

```
public void buttonClicked (View view)
{
  activityCallback.onButtonClick(arg1, arg2);
}
```

- All that remains is to modify the activity class so that it implements the ToolbarListener interface.

- For example:

```
public class MainActivity extends FragmentActivity
implements ToolbarFragment.ToolbarListener
{

  public void onButtonClick(String arg1, int arg2)
  {
    // Implement code for callback method
  }
  .
  .
}
```

- As we can see from the above code, the activity declares that it implements the ToolbarListener interface of the ToolbarFragment class and then proceeds to implement the onButtonClick() method as required by the interface.

- As outlined above, fragments provide a convenient mechanism for creating reusable modules of application functionality consisting of both sections of a user interface and the corresponding behavior.

- Once created, fragments can be embedded within activities.

- Having explored the overall theory of fragments in the previous chapter, the objective of this chapter is to create an example Android application using Android Studio designed to demonstrate the actual steps involved in both creating and using fragments, and also implementing communication between one fragment and another within an activity.

(continued)

About the Example Fragment Application

- The application created in this chapter will consist of a single activity and two fragments. The user interface for the first fragment will contain a toolbar of sorts consisting of an EditText view, a SeekBar and a Button, all contained within a ConstraintLayout view.

- The second fragment will consist solely of a TextView object, also contained within a ConstraintLayout view.

- The two fragments will be embedded within the main activity of the application and communication implemented such that when the button in the first fragment is pressed, the text entered into the EditText view will appear on the TextView of the second fragment using a font size dictated by the position of the SeekBar in the first fragment.

- Since this application is intended to work on earlier versions of Android, it will also be necessary to make use of the appropriate Android support library.

Creating the Example 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 Empty Activity template before clicking on the Next button.

- Enter FragmentExample into the Name field and specify edu.niu.your_Z-ID.fragmentexample 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.

Creating the First Fragment Layout

- The next step is to create the user interface for the first fragment that will be used within our activity.

- This user interface will consist of an XML layout file and a fragment class.

- While these could be added manually, it is quicker to ask Android Studio to create them for us.

- Within the project tool window, locate the app > java > edu.niu.your_Z-ID.fragmentexample entry and right click on it.

(continued)

- From the resulting menu, select the New > Fragment > Gallery... option to display the dialog shown in Figure 12-2 below:

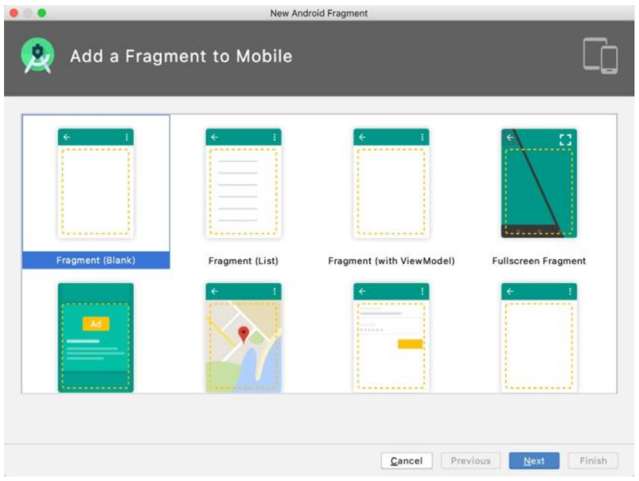
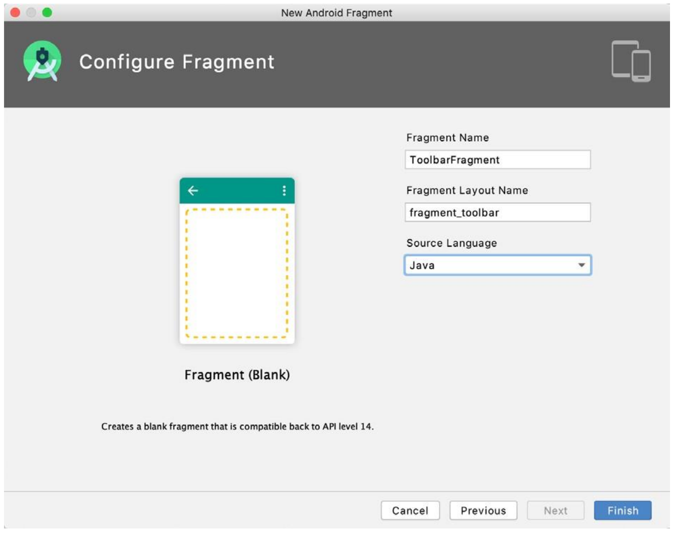


Figure 12-2

- Select the Fragment (Blank) template before clicking the Next button. On the subsequent screen, name the fragment ToolbarFragment with a layout file named fragment_toolbar:


New Android Fragment

Configure Fragment

Fragment Name: ToolbarFragment

Fragment Layout Name: fragment_toolbar

Source Language: Java

Fragment (Blank)

Creates a blank fragment that is compatible back to API level 14.

Figure 12-3

- Load the fragment_toolbar.xml file into the layout editor using Design mode, right-click on the FrameLayout entry in the Component Tree panel and select the Convert FrameLayout to ConstraintLayout menu option, accepting the default settings in the confirmation dialog.

- Select and delete the default TextView and add a Plain EditText, Seekbar and Button to the layout and change the view ids to editText1, button1 and seekBar1 respectively.

- Change the text on the button to read “Change Text”, extract the text to a string resource named change_text and remove the Name text from the EditText view.

- Finally, set the layout_width property of the Seekbar to match_constraint with margins set to 8dp on the left and right edges.

- Use the Infer constraints toolbar button to add any missing constraints, at which point the layout should match that shown in Figure 12-4 below:

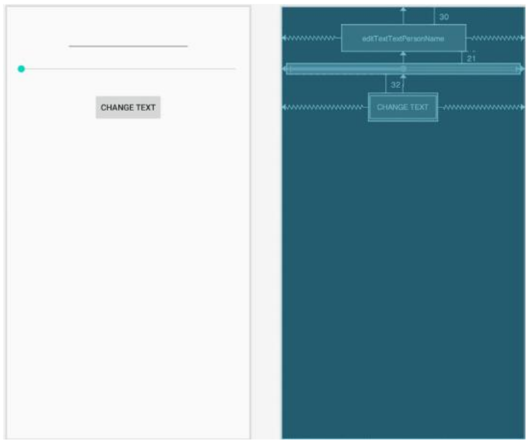


Figure 12-4

Adding the Second Fragment

- Repeating the steps used to create the toolbar fragment, add another empty fragment named TextFragment with a layout file named fragment_text.

- Once again, convert the FrameLayout container to a ConstraintLayout and remove the default TextView.

- Drag a drop a TextView widget from the palette and position it in the center of the layout, using the Infer constraints button to add any missing constraints.

- Change the id of the view to textView1, the text to read “Fragment Two” and modify the textAppearance attribute to Large.

- On completion, the layout should match that shown in Figure 12-5:

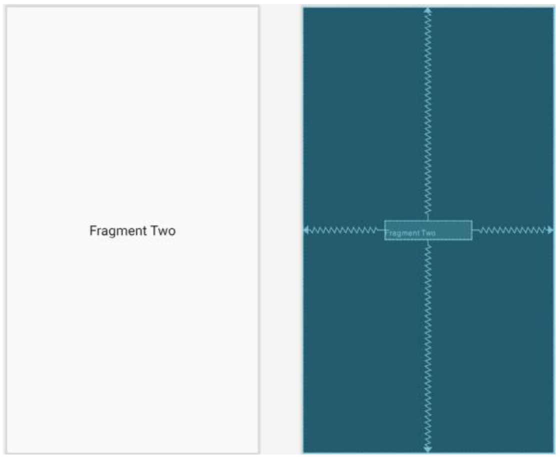


Figure 12-5

Adding the Fragments to the Activity

- The main activity for the application has associated with it an XML layout file named activity_main.xml.
- For the purposes of this example, the fragments will be added to the activity using the <fragment> element within this file.
- Using the Project tool window, navigate to the app > res > layout section of the FragmentExample project and double-click on the activity_main.xml file to load it into the Android Studio Layout Editor tool.
- With the Layout Editor tool in Design mode, select and delete the default TextView object from the layout and select the Common category in the palette.
- Drag the <fragment> component from the list of views and drop it onto the layout so that it is centered horizontally and positioned such that the dashed line appears indicating the top layout margin:

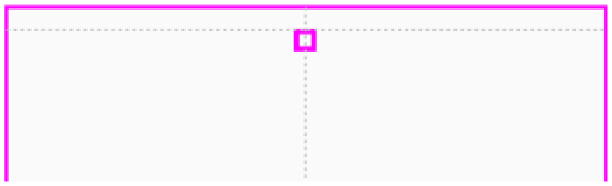
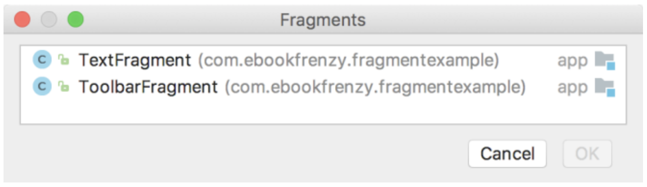


Figure 12-6

- On dropping the fragment onto the layout, a dialog will appear displaying a list of Fragments available within the current project as illustrated in Figure 12-7:

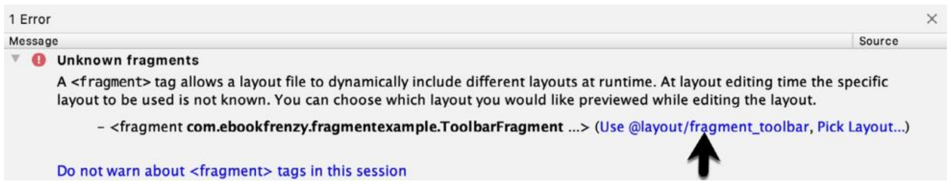


Fragments

TextFragment (com.ebookfrenzy.fragmentexample) app
ToolbarFragment (com.ebookfrenzy.fragmentexample) app

Figure 12-7

- Select the ToolbarFragment entry from the list and click on the OK button to dismiss the Fragments dialog.
- Once added, click on the red warning button in the top right-hand corner of the layout editor to display the warnings panel.
- An unknown fragments message (Figure 12-8) will be listed indicating that the Layout Editor tool needs to know which fragment to display during the preview session.
- Display the ToolbarFragment fragment by clicking on the Use @layout/toolbar_fragment link within the message:



1 Error

Message | Source

Unknown fragments

A <fragment> tag allows a layout file to dynamically include different layouts at runtime. At layout editing time the specific layout to be used is not known. You can choose which layout you would like previewed while editing the layout.

- <fragment com.ebookfrenzy.fragmentexample.ToolbarFragment ...> (Use @layout/fragment_toolbar, Pick Layout...)

Do not warn about <fragment> tags in this session

Figure 12-8

- With the fragment selected, change the layout_width property to match_constraint so that it occupies the full width of the screen.
- Click and drag another <fragment> entry from the panel and position it so that it is centered horizontally and located beneath the bottom edge of the first fragment.
- When prompted, select the TextFragment entry from the fragment dialog before clicking on the OK button.
- Display the error panel once again and click on the Use @layout/fragment_text option.
- Use the Infer constraints button to establish any missing layout constraints.
- Note that the fragments are now visible in the layout as demonstrated in Figure 12-9:

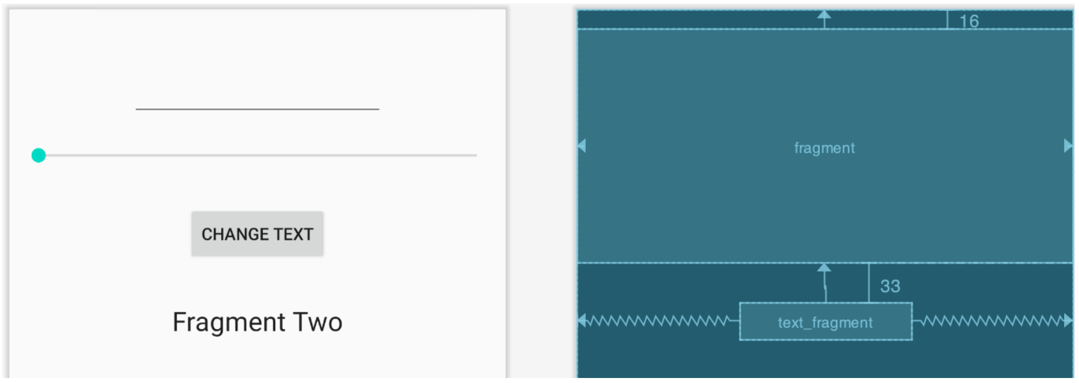


Figure 12-9

- Before proceeding to the next step, select the TextFragment instance in the layout and, within the Attributes tool window, change the ID of the fragment to text_fragment.

Making the Toolbar Fragment Talk to the Activity

- When the user touches the button in the toolbar fragment, the fragment class is going to need to get the text from the EditText view and the current value of the SeekBar and send them to the text fragment.
- As outlined in “An Introduction to Android Fragments”, fragments should not communicate with each other directly, instead using the activity in which they are embedded as an intermediary.
- The first step in this process is to make sure that the toolbar fragment responds to the button being clicked.
- We also need to implement some code to keep track of the value of the SeekBar view.
- For the purposes of this example, we will implement these listeners within the ToolbarFragment class.
- Select the ToolbarFragment.java file and modify it so that it reads as shown in the following listing:

```
package edu.niu.your_Z-ID.fragmentexample;
```

```
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.content.Context;
import android.widget.Button;
import android.widget.EditText;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

public class ToolbarFragment extends Fragment implements
  OnSeekBarChangeListener
{

  private static int seekvalue = 10;
  private static EditText edittext;

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState)
  {
    // Inflate the layout for this fragment
    View view =
      inflater.inflate(R.layout.fragment_toolbar, container, false);

    edittext = view.findViewById(R.id.editText1);
    final SeekBar seekbar = view.findViewById(R.id.seekBar1);
    seekbar.setOnSeekBarChangeListener(this);
    final Button button = view.findViewById(R.id.button1);

    button.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
        buttonClicked(v);
      }
    });
    return view;
  }

  public void buttonClicked (View view)
  {
  }

  @Override
  public void onProgressChanged(SeekBar seekBar, int progress,
                                boolean fromUser)
  {
    seekvalue = progress;
  }
```

```
    @Override
    public void onStartTrackingTouch(SeekBar arg0)
    {
    }

    @Override
    public void onStopTrackingTouch(SeekBar arg0)
    {
    }
  }
```

- Before moving on, we need to take some time to explain the above code changes.
- First, the class is declared as implementing the OnSeekBarChangeListener interface.
- This is because the user interface contains a SeekBar instance and the fragment needs to receive notifications when the user slides the bar to change the font size.
- Implementation of the OnSeekBarChangeListener interface requires that the onProgressChanged(), onStartTrackingTouch() and onStopTrackingTouch() methods be implemented.
- These methods have been implemented but only the onProgressChanged() method is actually required to perform a task, in this case storing the new value in a variable named seekvalue which has been declared at the start of the class.
- Also declared is a variable in which to store a reference to the EditText object.
- The onActivityCreated() method has been added to obtain references to the EditText, SeekBar and Button views in the layout.
- Once a reference to the button has been obtained it is used to set up an onClickListener on the button which is configured to call a method named buttonClicked() when a click event is detected.
- This method is also then implemented, though at this point it does not do anything.
- The next phase of this process is to set up the listener that will allow the fragment to call the activity when the button is clicked. This follows the mechanism outlined in the previous chapter:

```
public class ToolbarFragment extends Fragment implements
  OnSeekBarChangeListener
{

  private static int seekvalue = 10;
  private static EditText edittext;

  ToolbarListener activityCallback;

  public interface ToolbarListener
  {
    public void onButtonClick(int position, String text);
    }

    @Override
    public void onAttach(Context context)
    {
      super.onAttach(context);

      try
      {
        activityCallback = (ToolbarListener) context;
      }
      catch (ClassCastException e)
      {
        throw new ClassCastException(context.toString()
                                     + " must implement ToolbarListener");
      }
    }
    .
    .
    public void buttonClicked (View view)
    {
      activityCallback.onButtonClick(seekvalue,
                                     edittext.getText().toString());
    }
    .
    .
    .
  }
```

- The above implementation will result in a method named onButtonClick() belonging to the activity class being called when the button is clicked by the user.

- All that remains, therefore, is to declare that the activity class implements the newly created ToolbarListener interface and to implement the onButtonClick() method.

- Since the Android Support Library is being used for fragment support in earlier Android versions, the activity also needs to be changed to subclass from FragmentActivity instead of AppCompatActivity.

- Bringing these requirements together results in the following modified MainActivity.java file:

```
package edu.niu.your_Z-ID.fragmentexample;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentActivity;
import android.os.Bundle;

public class MainActivity extends FragmentActivity implements
  ToolbarFragment.ToolbarListener
{

  @Override
```

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

    public void onButtonClick(int fontsize, String text)
    {
    }
  }
```

- With the code changes as they currently stand, the toolbar fragment will detect when the button is clicked by the user and call a method on the activity passing through the content of the EditText field and the current setting of the SeekBar view.

- It is now the job of the activity to communicate with the Text Fragment and to pass along these values so that the fragment can update the TextView object accordingly.

Making the Activity Talk to the Text Fragment

- As outlined earlier in the chapter, an activity can communicate with a fragment by obtaining a reference to the fragment class instance and then calling public methods on the object.

- As such, within the TextFragment class we will now implement a public method named changeTextProperties() which takes as arguments an integer for the font size and a string for the new text to be displayed. The method will then use these values to modify the TextView object.

- Within the Android Studio editing panel, locate and modify the TextFragment.java file to add this new method and to add code to the onCreateView() method to obtain the ID of the TextView object:

```
package edu.niu.your_Z-ID.fragmentexample;

import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class TextFragment extends Fragment
{

  private static TextView textview;

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState)
  {
    View view = inflater.inflate(R.layout.fragment_text, container, false);
    textview = view.findViewById(R.id.textView1);
    return view;
    }

    public void changeTextProperties(int fontsize, String text)
    {
      textview.setTextSize(fontsize); textview.setText(text);
    }
  }
```

- When the TextFragment fragment was placed in the layout of the activity, it was given an ID of text_fragment.
- Using this ID, it is now possible for the activity to obtain a reference to the fragment instance and call the changeTextProperties() method on the object.
- Edit the MainActivity.java file and modify the onButtonClick() method as follows:

```
public void onButtonClick(int fontsize, String text)
{

  TextFragment textFragment = (TextFragment)
  getSupportFragmentManager().findFragmentById(R.id.text_fragment);

  textFragment.changeTextProperties(fontsize, text);
}
```

Testing the Application

- With the coding for this project now complete, the last remaining task is to run the application.
- When the application is launched, the main activity will start and will, in turn, create and display the two fragments.
- When the user touches the button in the toolbar fragment, the onButtonClick() method of the activity will be called by the toolbar fragment and passed the text from the EditText view and the current value of the SeekBar.
- The activity will then call the changeTextProperties() method of the second fragment, which will modify the TextView to reflect the new text and font size:

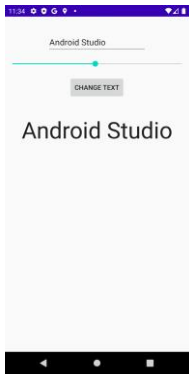


Figure 12-10

Summary

- Fragments provide a powerful mechanism for creating re-usable modules of user interface layout and application behavior, which, once created, can be embedded in activities.

- A fragment consists of a user interface layout file and a class.

- Fragments may be utilized in an activity either by adding the fragment to the activity's layout file, or by writing code to manage the fragments at runtime.

- Fragments added to an activity in code can be removed and replaced dynamically at runtime.

- All communication between fragments should be performed via the activity within which the fragments are embedded.

- Having covered the basics of fragments in this chapter, the next chapter will work through a tutorial designed to reinforce the techniques outlined in this chapter.

- The goal of this chapter was to work through the creation of an example project intended specifically to demonstrate the steps involved in using fragments within an Android application.

- Topics covered included the use of the Android Support Library for compatibility with Android versions predating the introduction of fragments, the inclusion of fragments within an activity layout and the implementation of inter-fragment communication.