- An area of user interface design that has not yet been covered in this book relates to the concept of menus within an Android application.

- Menus provide a mechanism for offering additional choices to the user beyond the view components that are present in the user interface layout.

- While there are a number of different menu systems available to the Android application developer, this chapter will focus on the more commonly used Overflow menu.

- The chapter will cover the creation of menus both manually via XML and visually using the Android Studio Layout Editor tool.

The Overflow Menu

- The overflow menu (also referred to as the options menu) is a menu that is accessible to the user from the device display and allows the developer to include other application options beyond those included in the user interface of the application.

- The location of the overflow menu is dependent upon the version of Android that is running on the device.

- With the Android 4.0 release and later, the overflow menu button is located in the top right-hand corner (Figure 14-1) in the action toolbar represented by the stack of three squares:

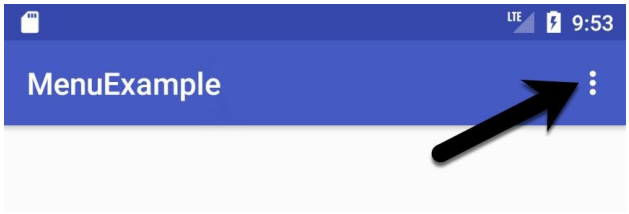

Figure 14-1

Creating an Overflow Menu

- The items in a menu can be declared within an XML file, which is then inflated and displayed to the user on demand.

- This involves the use of the <menu> element, containing an <item> sub-element for each menu item.

- The following XML, for example, defines a menu consisting of two menu items relating to color choices:

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

  <item
    android:id="@+id/menu_red" android:orderInCategory="1"
```

```
    app:showAsAction="never" android:title="@string/red_string"/>

    <item
      android:id="@+id/menu_green"
      android:orderInCategory="2" app:showAsAction="never"
      android:title="@string/green_string"/>

</menu>
```

- In the above XML, the android:orderInCategory property dictates the order in which the menu items will appear within the menu when it is displayed.

- The app:showAsAction property, on the other hand, controls the conditions under which the corresponding item appears as an item within the action bar itself.

- If set to if Room, for example, the item will appear in the action bar if there is enough room. Figure 14-2 shows the effect of setting this property to ifRoom for both menu items:

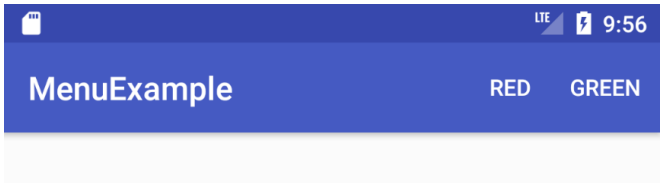

Figure 14-2

- This property should be used sparingly to avoid over cluttering the action bar.

- By default, a menu XML file is created by Android Studio when a new Android application project is created.

- This file is located in the app > res > menu project folder and contains a single menu item entitled "Settings":

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

    <item android:id="@+id/action_settings"
      android:title="@string/action_settings"
      android:orderInCategory="100" app:showAsAction="never" />
</menu>
```

- This menu is already configured to be displayed when the user selects the overflow menu on the user interface when the app is running, so simply modify this one to meet your needs.

Displaying an Overflow Menu

- An overflow menu is created by overriding the onCreateOptionsMenu() method of the corresponding activity and then inflating the menu's XML file.

- For example, the following code creates the menu contained within a menu XML file named menu_main:

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  getMenuInflater().inflate(R.menu.menu_main, menu);
  return true;
}
```

- As with the menu XML file, Android Studio will already have overridden this method in the main activity of a newly created Android application project.

- In the event that an overflow menu is not required in your activity, either remove or comment out this method.

Responding to Menu Item Selections

- Once a menu has been implemented, the question arises as to how the application receives notification when the user makes menu item selections.

- All that an activity needs to do to receive menu selection notifications is to override the onOptionsItemSelected() method.

- Passed as an argument to this method is a reference to the selected menu item.

- The getItemId() method may then be called on the item to obtain the ID which may, in turn, be used to identify which item was selected.

- For example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId())
  {
    case R.id.menu_red:   // Red item was selected
      return true;
    case R.id.menu_green:   // Green item was selected
      return true;
    default:
      return super.onOptionsItemSelected(item);
  }
}
```

Creating Checkable Item Groups

- In addition to configuring independent menu items, it is also possible to create groups of menu items.

- This is of particular use when creating checkable menu items whereby only one out of a number of choices can be selected at any one time.

- Menu items can be assigned to a group by wrapping them in the <group> tag.

- The group is declared as checkable using the android:checkableBehavior property, setting the value to either single, all or none.

- The following XML declares that two menu items make up a group wherein only one item may be selected at any given time:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">
  <group android:checkableBehavior="single">

    <item
      android:id="@+id/menu_red"
      android:title="@string/red_string"/>

    <item
      android:id="@+id/menu_green"
      android:title="@string/green_string"/>

  </group>
</menu>
```

- When a menu group is configured to be checkable, a small circle appears next to the item in the menu as illustrated in Figure 14-3.

- It is important to be aware that the setting and unsetting of this indicator does not take place automatically.

- It is, therefore, the responsibility of the application to check and uncheck the menu item.

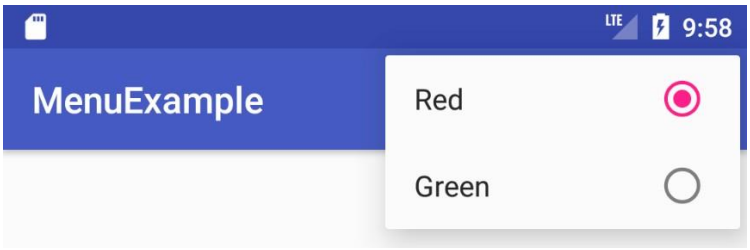

Figure 14-3

- Continuing the color example used previously in this chapter, this would be implemented as follows:

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId())
  {
```

```
      case R.id.menu_red:
        if (item.isChecked())
          item.setChecked(false);
        else
          item.setChecked(true);
        return true;
      case R.id.menu_green:
        if (item.isChecked())
          item.setChecked(false);
        else
          item.setChecked(true);
        return true;
      default:
        return super.onOptionsItemSelected(item);
    }
  }
```

Menus and the Android Studio Menu Editor

- Android Studio allows menus to be designed visually simply by loading the menu resource file into the Menu Editor tool, dragging and dropping menu elements from a palette and setting properties.
- This considerably eases the menu design process, though it is important to be aware that it is still necessary to write the code in the onOptionsItemSelected() method to implement the menu behavior.
- To visually design a menu, locate the menu resource file and double-click on it to load it into the Menu Editor tool.
- Figure 14-4, for example, shows the default menu resource file for a basic activity loaded into the Menu Editor in Design mode.
- The palette (A) contains items that can be added to the menu contained in the design area (C).
- The Component Tree (B) is a useful tool for identifying the hierarchical structure of the menu.
- The Attributes panel (D) contains a subset of common attributes for the currently selected item.
- The view all attributes link (E) may be used to access the full list of attributes.
- New elements may be added to the menu by dragging and dropping objects either onto the layout canvas or the Component Tree.
- When working with menus in the Layout Editor tool, it will sometimes be easier to drop the items onto the Component Tree since this provides greater control over where the item is placed within the tree.
- This is of particular use, for example, when adding items to a group.

(continued)

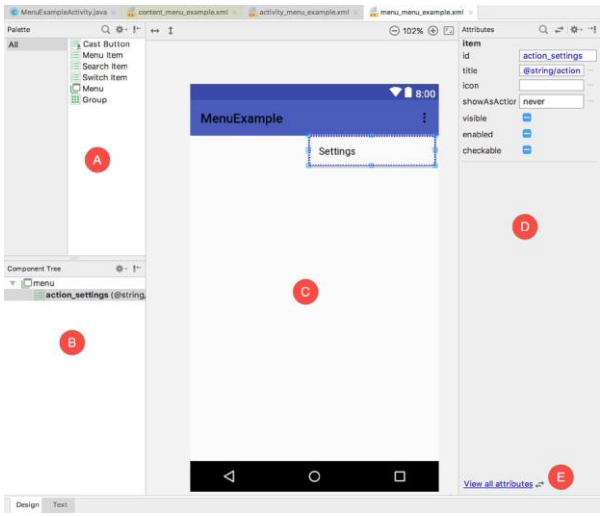

Figure 14-4

- Although the Menu Editor provides a visual approach to constructing menus, the underlying menu is still stored in XML format which may be viewed and edited manually by switching from Design to Code mode using the tab marked F in the above figure.

Creating the Example Project

- To see the overflow menu in action, 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 MenuExample into the Name field and specify edu.niu.your_Z-ID.menuexample 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.

- When the project has been created, navigate to the app > res > layout folder in the Project tool window and double-click on the content_main.xml file to load it into the Android Studio Menu Editor tool.

- Switch the tool to Design mode, select the ConstraintLayout from the Component Tree panel and enter layoutView into the ID field of the Attributes panel.

Designing the Menu

- Within the Project tool window, locate the project's app > res > menu > menu_main.xml file and double-click on it to load it into the Layout Editor tool.

- Switch to Design mode if necessary and select and delete the default Settings menu item added by Android Studio so that the menu currently has no items.

- From the palette, click and drag a menu group object onto the title bar of the layout canvas as highlighted in Figure 14-5:

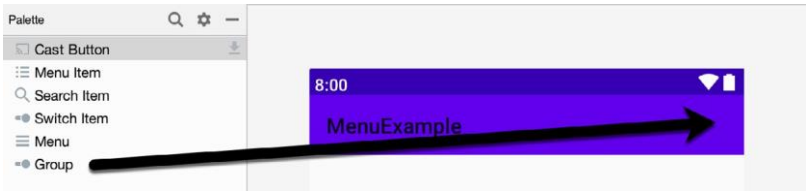

Figure 14-5

- Although the group item has been added, it will be invisible within the layout.

- To verify the presence of the element, refer to the Component Tree panel where the group will be listed as a child of the menu:

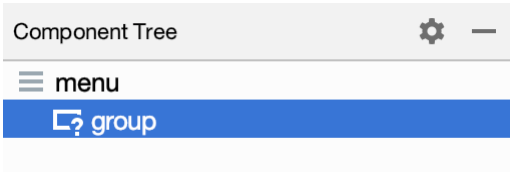

Figure 14-6

- Select the group entry in the Component Tree and, referring to the Attributes panel, set the checkableBehavior property to single so that only one group menu item can be selected at any one time:

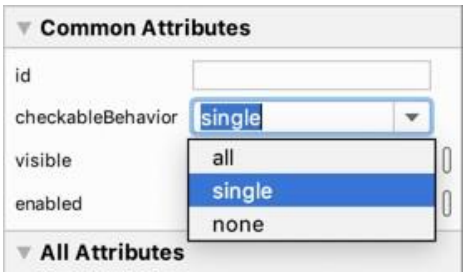

Figure 14-7

- Next, drag four Menu Item elements from the palette and drop them onto the group element in the Component Tree.

- Select the first item and use the Attributes panel to change the title to “Red” and the ID to menu_red:

Figure 14-8

- Repeat these steps for the remaining three menu items setting the titles to “Green”, “Yellow” and “Blue” with matching IDs of menu_green, menu_yellow and menu_blue.

- Use the warning buttons to the right of the menu items in the Component Tree panel to extract the strings to resources:

Figure 14-9

- On completion of these steps, the menu layout should match that shown in Figure 14-10 below:

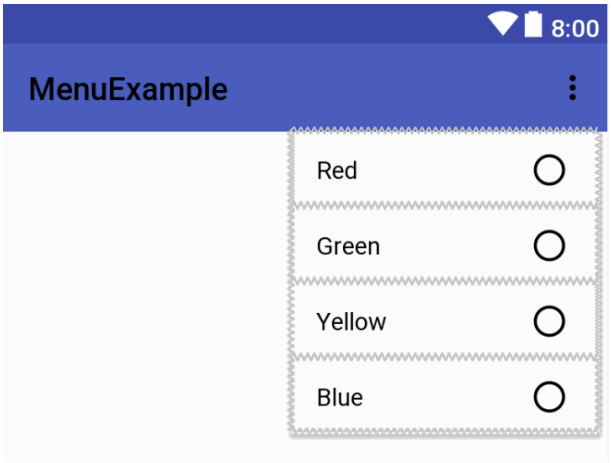

Figure 14-10

- Switch the Layout Editor tool to Code mode and review the XML representation of the menu which should match the following listing:

```
<menu 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"
  tools:context="edu.niu.your_Z-ID.menuexample.MainActivity">

  <group android:checkableBehavior="single">

    <item android:title="@string/red_string"
      android:id="@+id/menu_red" />

    <item android:title="@string/green_string"
      android:id="@+id/menu_green" />
```

```
    <item android:title="@string/yellow_string"
      android:id="@+id/menu_yellow" />

    <item android:title="@string/blue_string"
      android:id="@+id/menu_blue" />

  </group>
</menu>
```

Modifying the onOptionsItemSelected() Method

- When items are selected from the menu, the overridden onOptionsItemsSelected() method of the application's activity will be called.

- The role of this method will be to identify which item was selected and change the background color of the layout view to the corresponding color.

- Locate and double-click on the app > java > edu.niu.your_Z-ID.menuexample > MainActivity file and modify the method as follows:

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

import
com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.android.material.snackbar.Snackbar;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.view.View; import android.view.Menu;
import android.view.MenuItem;
import androidx.constraintlayout.widget.ConstraintLayout;

public class MainActivity extends AppCompatActivity
{
  .
  .
  @Override
  public boolean onOptionsItemSelected(MenuItem item)
  {

    ConstraintLayout mainLayout = findViewById(R.id.layoutView);

    switch (item.getItemId())
    {
      case R.id.menu_red:
        if (item.isChecked())
          item.setChecked(false);
        else
          item.setChecked(true);
```

```
            mainLayout.setBackgroundColor(android.graphics.Color.RED);
            return true;
          case R.id.menu_green:
            if (item.isChecked())
              item.setChecked(false);
            else
              item.setChecked(true);
            mainLayout.setBackgroundColor(android.graphics.Color.GREEN);
            return true;
          case R.id.menu_yellow:
            if (item.isChecked())
              item.setChecked(false);
            else
              item.setChecked(true);
              mainLayout.setBackgroundColor(android.graphics.Color.YELLOW);
              return true;
          case R.id.menu_blue:
            if (item.isChecked())
              item.setChecked(false);
            else
              item.setChecked(true);
            mainLayout.setBackgroundColor(android.graphics.Color.BLUE);
            return true;
          default:
            return super.onOptionsItemSelected(item);
        }
      }
      .
      .
}
```

Testing the Application

- Build and run the application on either an emulator or physical Android device.
- Using the overflow menu, select menu items and verify that the layout background color changes appropriately.

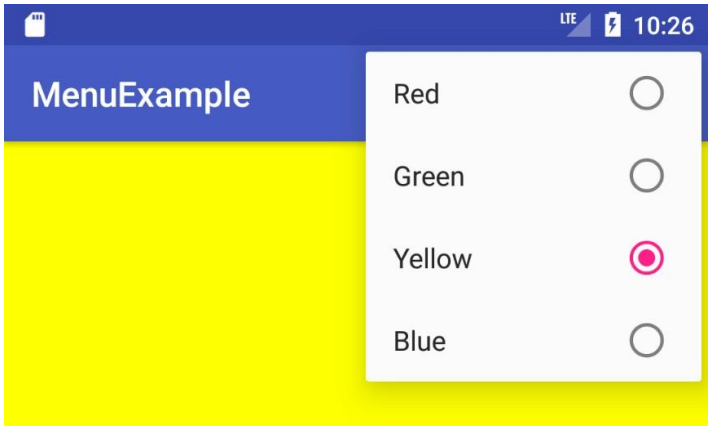

Figure 14-11

- Note that the currently selected color is displayed as the checked item in the menu.

The Material Design

- The overall appearance of the Android environment is defined by the principles of material design.

- Material design was created by the Android team at Google and dictates that the elements that make up the user interface of Android and the apps that run on it appear and behave in a certain way in terms of behavior, shadowing, animation and style.

- One of the tenets of the material design is that the elements of a user interface appear to have physical depth and a sense that items are constructed in layers of physical material.

- A button, for example, appears to be raised above the surface of the layout in which it resides through the use of shadowing effects.

- Pressing the button causes the button to flex and lift as though made of a thin material that ripples when released.

- Material design also dictates the layout and behavior of many standard user interface elements.

- A key example is the way in which the app bar located at the top of the screen should appear and the way in which it should behave in relation to scrolling activities taking place within the main content of the activity.
- In fact, material design covers a wide range of areas from recommended color styles to the way in which objects are animated.

- A full description of the material design concepts and guidelines can be found online at the following link and is recommended reading for all Android developers:

  https://www.google.com/design/spec/material-design/introduction.html

The Design Library

- Many of the building blocks needed to implement Android applications that adopt the principles of material design are contained within the Android Design Support Library.

- This library contains a collection of user interface components that can be included in Android applications to implement much of the look, feel and behavior of material design.

- Two of the components from this library, the floating action button and Snackbar, will be covered in this chapter, while others will be introduced in later chapters.

The Floating Action Button (FAB)

- The floating action button is a button which appears to float above the surface of the user interface of an app and is generally used to promote the most common action within a user interface screen.

- A floating action button might, for example, be placed on a screen to allow the user to add an entry to a list of contacts or to send an email from within the app.

- Figure 14-12, for example, highlights the floating action button that allows the user to add a new contact within the standard Android Contacts app:

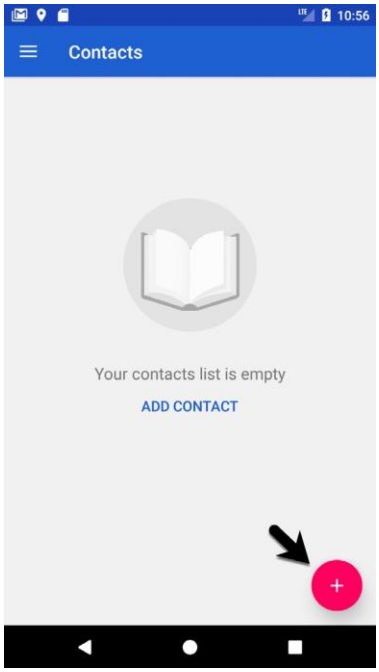

Figure 14-12

- To conform with the material design guidelines, there are a number of rules that should be followed when using floating action buttons.

- Floating action buttons must be circular and can be either 56 x 56dp (Default) or 40 x 40dp (Mini) in size.

- The button should be positioned a minimum of 16dp from the edge of the screen on phones and 24dp on desktops and tablet devices.

- Regardless of the size, the button must contain an interior icon that is 24x24dp in size and it is recommended that each user interface screen have only one floating action button.

- Floating action buttons can be animated or designed to morph into other items when touched.

- A floating action button could, for example, rotate when tapped or morph into another element such as a toolbar or panel listing related actions.

The Snackbar

- The Snackbar component provides a way to present the user with information in the form of a panel that appears at the bottom of the screen as shown in Figure 14-13.

- Snackbar instances contain a brief text message and an optional action button which will perform a task when tapped by the user.

- Once displayed, a Snackbar will either timeout automatically or can be removed manually by the user via a swiping action. During the appearance of the Snackbar the app will continue to function and respond to user interactions in the normal manner.

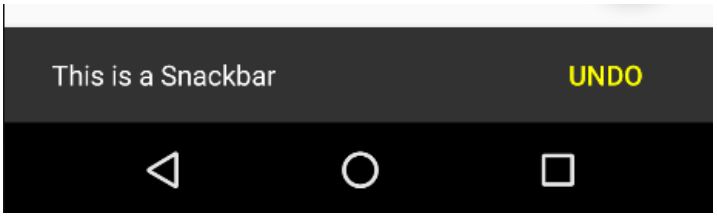

Figure 14-13

- In the remainder of this chapter an example application will be created that makes use of the basic features of the floating action button and Snackbar to add entries to a list of items.

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 Basic Activity template before clicking on the Next button.

- Enter FabExample into the Name field and specify edu.niu.your_Z-ID.fabexample 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.

Reviewing the Project

- Since the Basic Activity template was selected, the activity contains two layout files.

- The activity_main.xml file consists of a CoordinatorLayout manager containing entries for an app bar, a toolbar and a floating action button.

- The content_main.xml file represents the layout of the content area of the activity and contains a NavHostFragment instance.

- This file is embedded into the activity_main.xml file via the following include directive:

```
<include layout="@layout/content_main" />
```

- The floating action button element within the activity_main.xml file reads as follows:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
  android:id="@+id/fab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom|end"
  android:layout_margin="@dimen/fab_margin"
  app:srcCompat="@android:drawable/ic_dialog_email" />
```

- This declares that the button is to appear in the bottom right-hand corner of the screen with margins represented by the fab_margin identifier in the values/dimens.xml file (which in this case is set to 16dp).

- The XML further declares that the interior icon for the button is to take the form of the standard drawable built-in email icon.

- The blank template has also configured the floating action button to display a Snackbar instance when tapped by the user.

- The code to implement this can be found in the onCreate() method of the MainActivity.java file and reads as follows:

```
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 code obtains a reference to the floating action button via the button's ID and adds to it an onClickListener handler to be called when the button is tapped.
- This method simply displays a Snackbar instance configured with a message but no actions.

- When the project is compiled and run the floating action button will appear at the bottom of the screen as shown in Figure 14-14:

Figure 14-14

- Tapping the floating action button will trigger the onClickListener handler method causing the Snackbar to appear at the bottom of the screen:

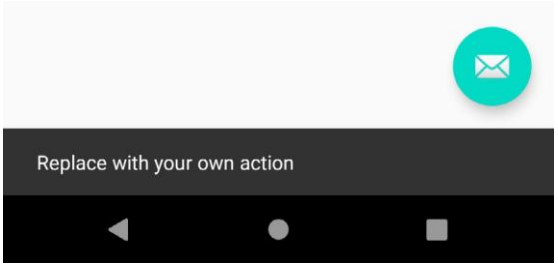

Figure 14-15

- When the Snackbar appears on a narrower device (as is the case in Figure 14-15 above) note that the floating action button is moved up to make room for the Snackbar to appear.

- This is handled for us automatically by the CoordinatorLayout container in the activity_main.xml layout resource file.

Removing Navigation Features

- As outlined before, the Basic Activity template contains multiple fragments and buttons to navigate from one fragment to the other.

- For the purposes of this tutorial, these features are unnecessary and will cause problems later if not removed.

- Before moving ahead with the tutorial, modify the project as follows:

1. Within the Project tool window, navigate to and double-click on the app > res > navigation > nav_graph.
2. xml file to load it into the navigation editor.

3. Within the editor, select the SecondFragment entry in the Destinations panel and tap the keyboard delete key to remove it from the graph.

4. Locate and delete the SecondFragment.java (app > java > <package name> > SecondFragment) and fragment_second.xml (app > res > layout > fragment_second.xml) files.

5. Locate the FirstFragment.java file, double click on it to load it into the editor and remove the code from the onViewCreated() method so that it reads as follows:

```
public void onViewCreated(@NonNull View view, Bundle savedInstanceState)
{
  super.onViewCreated(view, savedInstanceState);

  view.findViewById(R.id.button_first).setOnClickListener(new View.
    OnClickListener()
  {
    @Override
    public void onClick(View view)
    {
      NavHostFragment.findNavController(FirstFragment.this)
          .navigate(R.id.action_FirstFragment_to_SecondFragment);
    }
  });
}
```

Changing the Floating Action Button

- Since the objective of this example is to configure the floating action button to add entries to a list, the email icon currently displayed on the button needs to be changed to something more indicative of the action being performed.

- The icon that will be used for the button is named ic_add_entry.png 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 this image in the file system navigator for your operating system and copy the image file.

- Right-click on the app > res > drawable entry in the Project tool window and select Paste from the menu to add the file to the folder:

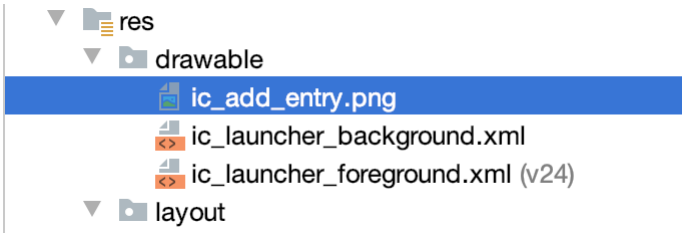

Figure 14-16

- Next, edit the activity_main.xml file and change the image source for the icon from @android:drawable/ic_ dialog_email to @drawable/ic_add_entry as follows:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
  android:id="@+id/fab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom|end"
  android:layout_margin="@dimen/fab_margin"
  app:srcCompat="@drawable/ic_add_entry" />
```

- Within the layout preview, the interior icon for the button will have changed to a plus sign.

- The background color of the floating action button is defined by the colorAccent property of the prevailing theme used by the application.

- The color assigned to this value is declared in the colors.xml file located under app > res values in the Project tool window.

- Edit this file and change the colorAccent property to a different color value:

```
<?xml version="1.0" encoding="utf-8"?>
  <resources>
    <color name="colorPrimary">@color/colorPrimary</color>
    <color name="colorPrimaryDark">@color/colorPrimary</color>
    <color name="colorAccent">#FFDE03</color>
  </resources>
```

- Return to the activity_main.xml file and verify that the floating action button now appears with a yellow background.

Adding the ListView to the Content Layout

- The next step in this tutorial is to add the ListView instance to the fragment_first.xml file.
- The ListView class provides a way to display items in a list format and can be found in the Legacy section of the Layout Editor tool palette.
- Load the fragment_first.xml file into the Layout Editor tool, select Design mode if necessary, and select and delete the default TextView and Button objects.
- Locate the ListView object in the Legacy category of the palette and, with autoconnect mode enabled, drag and drop it onto the center of the layout canvas.
- Select the ListView object and change the ID to listView within the Attributes tool window.
- The Layout Editor should have sized the ListView to fill the entire container and established constraints on all four edges as illustrated in Figure 14-17:

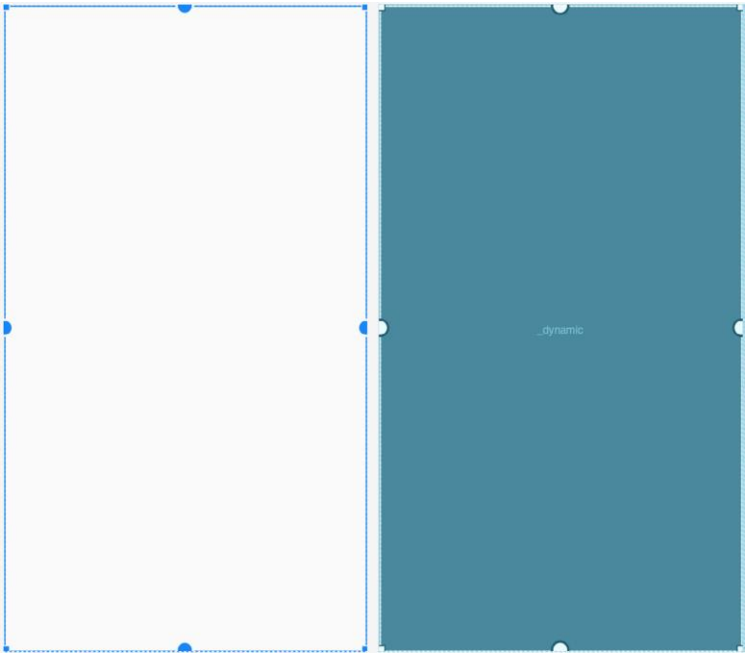

Figure 14-17

- With the ListView object selected, use the Attributes tool window to set the layout_width and layout_height properties to 0dp (match constraint).

Adding Items to the ListView

- Each time the floating action button is tapped by the user, a new item will be added to the ListView in the form of the prevailing time and date.
- To achieve this, some changes need to be made to the MainActivity.java file.

- Begin by implementing the onStart() method to obtain a reference to the ListView instance and to initialize an adapter instance to allow us to add items to the list in the form of an array:

```
import android.os.Bundle;
import
  com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.android.material.snackbar.Snackbar;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity
{

  ArrayList<String> listItems = new ArrayList<String>();
  ArrayAdapter<String> adapter;
  private ListView myListView;
  .
  .
  @Override
  protected void onStart()
  {
    super.onStart();
    myListView = findViewById(R.id.listView);

    adapter = new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_1, listItems);
    myListView.setAdapter(adapter);
  }
  .
  .
}
```

- Note that the onStart() lifecycle method is used here instead of the usual onCreate() method.

- This is because the fragment containing the ListView object will not have been initialized at point that the onCreate() method is called resulting in the app to crashing.

- The ListView needs an array of items to display, an adapter to manage the items in that array and a layout definition to dictate how items are to be presented to the user.

- In the above code changes, the items are stored in an ArrayList instance assigned to an adapter that takes the form of an ArrayAdapter.

- The items added to the list will be displayed in the ListView using the simple_list_ item_1 layout, a built-in layout that is provided with Android to display simple string based items in a ListView instance.

- Next, edit the onClickListener code for the floating action button to display a different message in the Snackbar and to call a method to add an item to the list:

```
FloatingActionButton fab = findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener()
{
  @Override
  public void onClick(View view)
  {
    addListItem();
    Snackbar.make(view, "Item added to list", Snackbar.LENGTH_LONG)
      .setAction("Action", null).show();

  }
});
```

- Remaining within the MainActivity.java file, add the addListItem() method as follows:

```
package edu.niu.your_Z-ID.fabexample;
.
.
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class MainActivity extends AppCompatActivity
{
  .
  .
  private void addListItem()
  {

    SimpleDateFormat dateformat =
      new SimpleDateFormat("HH:mm:ss MM/dd/yyyy", Locale.US);
    listItems.add(dateformat.format(new Date()));
    adapter.notifyDataSetChanged();
  }
  .
  .
}
```

- The code in the addListItem() method identifies and formats the current date and time and adds it to the list items array.

- The array adapter assigned to the ListView is then notified that the list data has changed, causing the ListView to update to display the latest list items.

- Compile and run the app and test that tapping the floating action button adds new time and date entries to the ListView, displaying the Snackbar each time as shown in Figure 14-18:

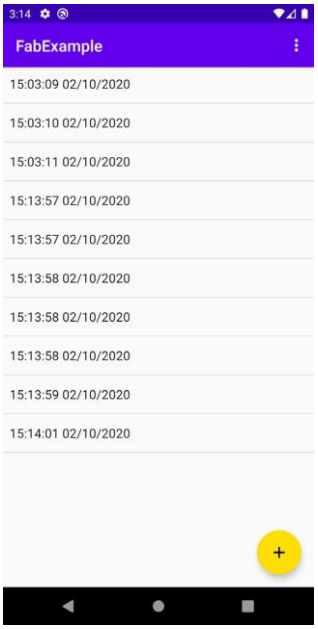

Figure 14-18

Adding an Action to the Snackbar

- The final task in this project is to add an action to the Snackbar that allows the user to undo the most recent addition to the list.

- Edit the MainActivity.java file and modify the Snackbar creation code to add an action titled "Undo" configured with an onClickListener named undoOnClickListener:

```
fab.setOnClickListener(new View.OnClickListener()
{
  @Override
  public void onClick(View view)
  {
    addListItem();
    Snackbar.make(view, "Item added to list", Snackbar.LENGTH_LONG)
     .setAction("Undo", undoOnClickListener).show();

  }
});
```

- Within the MainActivity.java file add the listener handler:

```
View.OnClickListener undoOnClickListener = new View.OnClickListener()
{
  @Override
  public void onClick(View view)
  {
    listItems.remove(listItems.size() -1);
    adapter.notifyDataSetChanged();
    Snackbar.make(view, "Item removed", Snackbar.LENGTH_LONG)
```

```
        .setAction("Action", null).show();
    }
};
```

- The code in the onClick method identifies the location of the last item in the list array and removes it from the list before triggering the list view to perform an update.

- A new Snackbar is then displayed indicating that the last item has been removed from the list.

- Run the app once again and add some items to the list.

- On the final addition, tap the Undo button in the Snackbar (Figure 14-19) to remove the last item from the list:

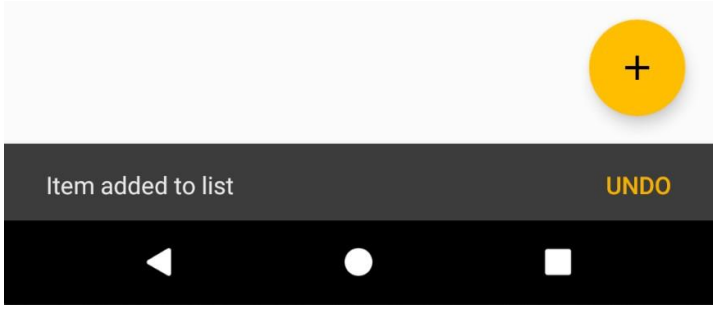

Figure 14-19

- It is also worth noting that the Undo button appears using the same color assigned to the accentColor property via the Theme Editor earlier in the chapter.

Summary

- The Android overflow menu is accessed from the far right of the actions toolbar at the top of the display of the running app.

- This menu provides a location for applications to provide additional options to the user.

- The structure of the menu is most easily defined within an XML file and the application activity receives notifications of menu item selections by overriding and implementing the onOptionsItemSelected() method.

- One of the objectives of this chapter is to provide an overview of the concepts of material design.

- Originally introduced as part of Android 5.0, material design is a set of design guidelines that dictate how the Android user interface, and that of the apps running on Android, appear and behave.

- As part of the implementation of the material design concepts, Google also introduced the Android Design Support Library.

- This library contains a number of different components that allow many of the key features of material design to be built into Android applications.

- Two of these components, the floating action button and Snackbar, will also be covered in this chapter prior to introducing many of the other components in subsequent chapters.

- This chapter has provided a general overview of material design, the floating action button and Snackbar before working through an example project that makes use of these features.

- Both the floating action button and the Snackbar are part of the material design approach to user interface implementation in Android.

- The floating action button provides a way to promote the most common action within a particular screen of an Android application.

- The Snackbar provides a way for an application to both present information to the user and also allow the user to take action upon it.