Exercise 1-2 Open an existing project and run it

Open a project and review its code

1. Start Android Studio.
2. Open the project that's stored in this directory:

   `\murach\book_apps\ch10_NewsReader`

3. Open all three layouts in the res\layout directory. Click the Design tab to view these layouts the graphical editor. In addition, click the Text tab to view the XML for these layouts. Note that the activity_items layout defines a screen for multiple items, and the activity_item layout defines a screen for a single item.
4. Open the Java classes for both activities (ItemsActivity and ItemActivity). Review this code. Note how each Java class works with the widgets displayed in the corresponding layout.
5. Open the other Java classes for the app. Note that these classes rely mostly on classes from the Java API, not classes from the Android API.

Run the app on a device

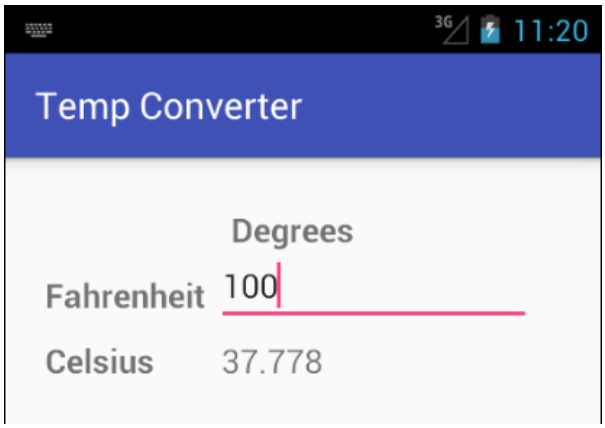
6. Run the app and use the Choose Device dialog to select the device that's connected to your computer.
7. Test the News Reader app by scrolling through the list of news items. Then, click on an item to display more details about it.

Run the app on an emulator (optional)

8. Run the app and use the Choose Device dialog to select the emulator that you want to use. If necessary, you can use this dialog to start the emulator. In that case, it may take a long time for the emulator to launch, so be patient!
9. Test the News Reader app by scrolling through the list of news items. Then, click on an item to display more details about it.

Exercise 2-3 Create the user interface for the Temp Converter app

In this exercise, you'll create the user interface for an app that converts temperature from degrees Fahrenheit to degrees Celsius. When you're done, the app should look like this:



Create the project

1. Start Android Studio.
2. Create a project for an Android app named Temp Converter and store it in this directory:

   `\murach\android\ex_starts`

   This project should be stored in a package named com.*lastname*.tempconverter, and it should be based on the Empty Activity template.
3. If you have an Android device, run the app on that device. Otherwise, run it in an emulator. This should display a message that says "Hello world!" in the center of the screen.

Create the user interface

4. Navigate to the res\layout directory and open the layout for the activity. If necessary, click the Design tab to display the graphical editor.
5. Delete the TextView widget that displays the "Hello world!" message.
6. Add the four TextView widgets and one EditText widget to the layout. Set the id and text properties of each widget immediately after you add the widget. When you're done, the user interface should have the widgets and text shown above. However, these widgets may look different since you haven't set their properties yet.
7. Set the textSize property for all widgets to 18sp.
8. Set the textStyle property for all widgets that label other widgets to bold.
9. Test the user interface by running the app on a physical device or an emulator. At this point, the app should allow you to enter the degrees in Fahrenheit. However, it doesn't yet convert those degrees to Celsius.

Exercise 3-3 Modify the Tip Calculator app

In this exercise, you'll modify the Tip Calculator application that's presented in this chapter.

Test the lifecycle of an activity

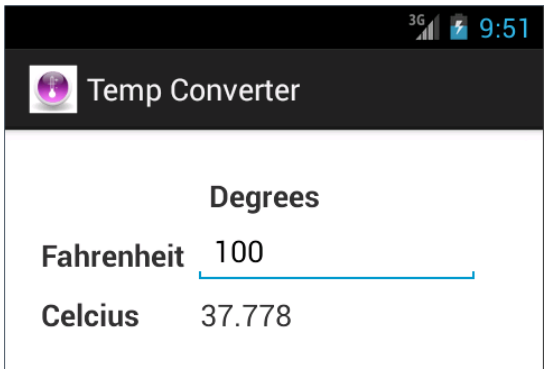
1. Start Android Studio and open the project named ch03_ex3_TipCalculator.
2. Test this app with valid data to see how it works.
3. Enter a bill amount and change the tip percent. Then, change orientation of the screen. The app should lose the value of the tip percent.
4. Enter a bill amount and change the tip percent. Then, navigate away from this app and return to it. The app should lose the string in the editable text view for the bill amount and the value of the tip percent.
5. Add an onPause method that saves the data for the bill amount and the tip percent instance variable. To do that, you can add an instance variable for the bill amount string.
6. Modify the onResume method so it restores the data that you saved.
7. Repeat steps 3 and 4 to make sure the app saves and restores its data.

Add a widget and handle its Click event

8. Open the layout for the activity and add a Reset button to the lower right corner.
9. Open the class for the activity and modify the code so clicking on the Reset button clears any text from the EditText widget and resets the tip percent to 15%.
10. Test this change to make sure it works correctly.

Exercise 3-4 Write the Java code for the Temp Converter app

In this exercise, you'll write the Java code for an app that converts temperature from degrees Fahrenheit to degrees Celsius. When you're done, the app should look like this:



Open the app and test it

1. Start Android Studio and open the project named ch03_ex4_TempConverter.
2. Run this project and test the app with a valid subtotal like 100. The app should accept this input, but it shouldn't perform any calculations or display any results.

Write the Java code

3. Open the Java class for the only activity of this app.
4. Use the onCreate method to get references to the EditText widget and the TextView widget that displays the degrees in Celsius.
5. Create an event handler for the EditorAction event for the EditText widget. The event handler should calculate and display the degrees in Celsius when the Done key is pressed on the soft keyboard. The formula for converting temperatures from Fahrenheit to Celsius is:

```
c = (f-32) * 5/9
```

6. Test the app. At this point, it should make the calculation correctly. However, it will lose its data if you change orientation or navigate away from the app.
7. Override the onPause method so it saves a string for the degrees Fahrenheit and Celsius. Then, modify the onResume method so it gets these strings and sets them on the appropriate widgets.
8. Test the app again. This time, the app should always remember its data even if you change orientation or navigate away from the app and return to it.

Exercise 4-2 Test and debug an app

In this exercise, you'll use Android Studio to find and fix four syntax errors.

Use LogCat logging

1. Start Android Studion and open the project named ch04_ex2_InvoiceTotal.
2. Add a logging statement to the end of the onPause method that indicates that the user data has been saved. This message should include the value of the subtotalString variable.
3. Add a logging statement to the end of the onResume method that indicates that the user data has been restored. This message should include the value of the subtotalString variable.
4. Run the app and change the orientation. In the LogCat view, you should be able to clearly see when the app saves and restores the subtotalString variable. If this step doesn't work correctly on a physical device, you may need to troubleshoot the problem for that device. Or, you can try this step on an emulator instead.

Use the debugger to step through all code

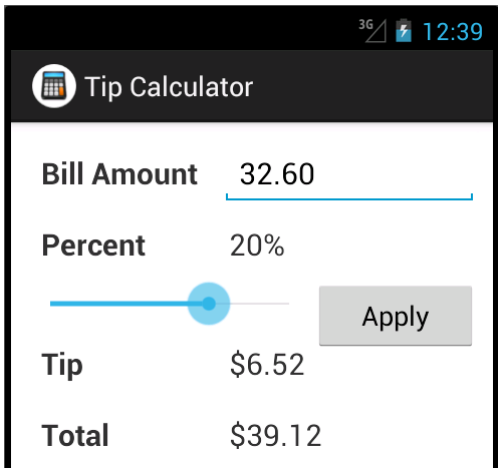
5. Set a breakpoint on the line in the onResume method that calls the calculateAndDisplay method.
6. Run the app in debug mode. Code execution should stop at the breakpoint.
7. Use the Step Into button to step through the code. Note that this allows you to step into the calculateAndDisplay method. At each step, note the values that are displayed in the Variables window.
8. When you finish stepping through the code, the device or emulator should be waiting for you to enter a subtotal. So, enter a new subtotal and press the Done button.
9. Use the Step Over button to step through the code. Note that this steps over the calculateAndDisplay method and returns control to the device or emulator.
10. Stop the debugger and remove the breakpoint.

Use the debugger to find and fix a bug

11. Run the app and test it with a subtotal of 300. Note that this doesn't yield a correct calculation. The discount percent should be 20% for amounts over 200.
12. In the calculateAndDisplay method, set a breakpoint somewhere before the if/else statement that gets the discount percent.
13. Run the app in debug mode. The app should stop at the breakpoint. Click the Resume button, switch to the device or emulator, enter a value of 300, and click the Done button.
14. Step through the if/else statement. As you do, you should be able to figure out what's causing the bug.
15. Find and fix the bug!
16. Stop the debugger and remove the breakpoint.

Exercise 5-3 Add a seek bar

In this exercise, you'll modify the Tip Calculator app so that it uses a seek bar instead of two buttons to set the tip percent. When you're done, a test run should look like this:



Test the app

1. Start Android Studio and open the project named ch05_ex3_TipCalculator.
2. Test this app to see how it works. Note that the Apply button doesn't do anything.

Add a seek bar

3. Open the layout for the activity and delete the two buttons that set the tip percent.
4. Create a new row below the "Percent" row that consists of a seek bar followed by the Apply button as shown above.
5. The seek bar should have a maximum value of 30 and a default progress of 15.
6. Open the class for the activity and delete all code related two the two buttons that have been deleted.
7. Modify the calculateAndDisplay method so it gets the correct tip percent from the SeekBar widget and displays the correct tip percent on the related TextView widget.
8. Test this change to make sure it works correctly. Note that you must click the Apply button to display the new tip percent that has been set by the seek bar. Although this seek bar should work, it doesn't provide a responsive user interface. That's why the next chapter shows how to handle the events of a seek bar to provide a more responsive user interface.

Exercise 6-3 Use anonymous inner classes for event listeners

In this exercise, you'll modify the Tip Calculator app that's presented in this chapter so it uses anonymous inner classes for event listeners.

1. Start Android Studio and open the project named ch06_ex3_TipCalculator.
2. Open the class for the activity and review its code. Note that the activity class implements five listener interfaces.
3. Test the app to make sure the EditText, SeekBar, RadioGroup, and Spinner widgets all work correctly.
4. Modify the code so it uses an anonymous inner class for the OnEditorActionListener. To do that, modify the declaration for the class so it doesn't implement this interface. Then, modify the argument for the method that sets the listener for the EditText widget so that the argument creates the listener object from an anonymous inner class.
5. Test the app to make sure the EditText widget still works correctly.
6. Repeat steps 4 and 5 for the listeners for the SeekBar, RadioGroup, and Spinner widgets.

Exercise 7-4 Use styles and themes with new widget types

In this exercise, you'll modify the Tip Calculator app presented in this chapter so it uses styles and themes to format some of the widgets that were introduced in chapter 5.

Use styles with some new widgets

1. Start Android Studio and open the project named ch07_ex4_TipCalculator.
2. Open the layout for the activity and review its XML. Note that some widgets use a style to apply formatting while other widgets don't use styles.
3. Open the styles.xml file and review its XML.
4. In the layout, remove the padding, textSize, and textStyle attributes from all TextView widgets below the separator line. Then, apply the TextView and TextView.Label styles to restore the formatting.
5. In the styles.xml file, create a custom style named PaddedWidget that adds 10dp of padding.
6. In the layout, remove the padding attribute from the SeekBar and RadioGroup widgets. Then, apply the PaddedWidget style to these widgets to restore this padding.
7. In the styles.xml file, change the padding attributes for the TextView.Label and PaddedWidget styles to 5dp. This should adjust the padding for all widgets on the layout.
8. In the styles.xml file, change the textSize attribute of the TextView style to 24sp. This should change the text size for most of the widgets on the layout.

Use themes to format radio buttons

9. Run the app on a device or emulator that uses an API from 15 to 20. Note that this device uses the Holo theme. Note also that the text size for the radio buttons is smaller than the text size for most other widgets.
10. In the graphical layout editor, click on a radio button and view its textAppearance property. It should not be set.
11. Open the styles.xml file in the values directory. Note that this file can contain customizations that apply to all APIs or to just APIs 15 to 20.
12. Add a style that has Android's TextAppearance style as its parent style and modify it so it sets the textSize to 24sp.
13. Customize this theme so it uses the textAppearance attribute to point to the style that you just created. This should change the text size for all three radio buttons for APIs 15 to 20.

Use themes to format spinner items

14. Run the app on device or emulator that uses an API from 15 to 20. Note that the text size for the spinner items is smaller than the text size for most other widgets.
15. Open the styles.xml file in the values directory.
16. Add a style that uses the Widget.TextView.SpinnerItem as the parent style.
17. Modify this style so it sets the text size to 24sp.
18. Customize the theme so its spinnerItemStyle attribute points to the new style. This should change the text size for the spinner item for all APIs 15 to 20.

Customize API 21 and later

19. Open the styles.xml file in the values-v21 directory. Note that it specifies the Material theme, not the Holo theme.
20. Run the app on a device or emulator that uses API 21 or later. Note that this uses the Material theme. Note also that the text size for the radio buttons and the spinner item is smaller than the text size for most other widgets.
21. Open the styles.xml file in the values directory. Move the attributes that customize the radio buttons and the spinner item from the customizations for APIs 15-20 to the customizations for all APIs. This should change the text size for the radio buttons and the spinner item for all APIs, including APIs 21 and later.

Exercise 7-5 Use colors

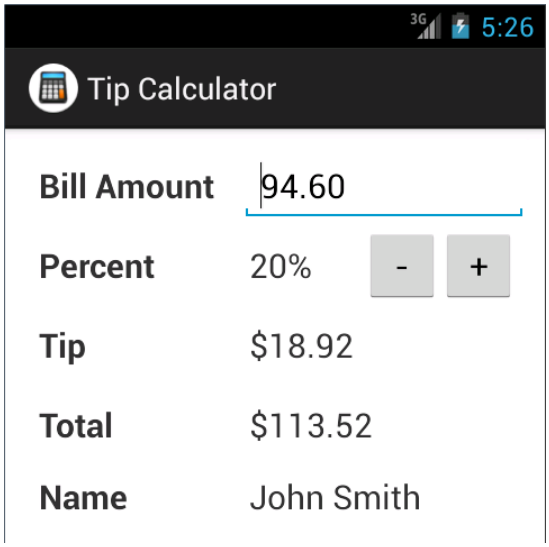
In this exercise, you'll modify the Tip Calculator app presented in this chapter so it uses custom colors.

1. Start Android Studio and open the project named ch07_ex5_TipCalculator.
2. Open the colors.xml file in the values directory. This file defines several custom colors.
3. Open the styles.xml file in the values directory. This file customizes the default theme.
4. Open the layout for the app in the graphical editor. This layout should use the colors from Android's built-in themes.
5. Modify the styles.xml file so the theme named AppTheme sets the windowBackground attribute of the theme to the custom color named background.
6. View the layout in the graphical editor. The background color for the window should now be light green.
7. Modify the styles.xml file so the theme named AppTheme sets the attributes named textColorPrimary and textColorSecondary to the custom color named white.
8. View the layout in the graphical editor. The text color for the TextView and TextEdit widgets should now be white, but the text color for the Buttons should still be black.
9. Modify the styles.xml file so the style named TextView.Label sets the textColor attribute to the color named black.
10. View the layout in the Graphical Layout editor. The text color for the labels should now be black.

Exercise 8-4 Use an EditTextPreference element

In this exercise, you'll modify the Tip Calculator app presented in this chapter so it uses an EditTextPreference element. Although this type of preference wasn't described in this chapter, it works similarly to the CheckBoxPreference element, and you can look it up in the documentation for the API if necessary.

When you're done, a test run should look like this:



1. Start Android Studio and open the project named ch08_ex4_TipCalculator.
2. Open the preferences.xml file in the xml directory. Add an EditTextPreference element for a setting named Name. This setting should allow the user to enter his or her full name.
3. Run the app and use the Settings activity to enter your name.
4. Open the layout of the activity. Modify the layout, so it includes a fifth row that can display a name.
5. Open the class for the Tip Calculator activity. Modify this code so it gets the name from the preferences and displays that name in the fifth row.

Exercise 9-4 Convert activites to fragments

In this exercise, you'll convert the News Reader app presented in chapter 10 so it uses fragments to allow this app to work better on devices with large screens. Although this app uses some new coding skills that you haven't been introduced to yet, you can ignore them for now and focus on converting this app's activities to fragments.

Test the app

1. Start Android Studio and open the project named ch09_ex4_NewsReader.
2. Run the app and test it. Note that it uses two activities to allow you to view a list of items or a single item.

Convert the Items activity so it uses a fragment

3. Make a copy of the activity_items layout and name it fragment_items.
4. Add the source code for a fragment named ItemsFragment. You can move most of the code that provides the functionality from the ItemsActivity class. To get this code to work correctly, you can use the getActivity method instead of the *this* keyword whenever you need to get a reference to the current activity.
5. Modify the activity_items layout so it displays the fragment_items layout.
6. Modify the ItemsActivity class so it only displays the activity_items layout.
7. Test the app to make sure it works. At this point, it should work the same as it did in step 2.

Convert the Item activity so it uses a fragment

8. Make a copy of the activity_item layout and name it fragment_item.
9. Add the source code for a fragment named ItemFragment. You can move most of the code that provides the functionality from the ItemActivity class. To get this code to work correctly, you can use the getActivity method instead of the *this* keyword whenever you need to get a reference to the current activity.
10. Modify the activity_item layout so it displays the fragment_item layout.
11. Modify the ItemActivity class so it only displays the activity_item layout.
12. Test the app to make sure it works. At this point, it should work the same as it did in step 2.

Exercise 10-4 Move a task to an asychronous thread

In this exercise, you'll modify an app that downloads an image file from the Internet, writes that image to a file, and reads that image from the file.

Review and test the app

1. Start Android Studio and open the project named ch10_ex4_TestAsync.
2. Review the code for this app. Note that it uses an asynchronous task to download an image file from the Internet.
3. Run the app. When it starts, it should briefly display the standard Android image. Then, it should download another image from the Internet, write it to a file, read that file, and display the image in an ImageView widget.

Move a task from the UI thread to another thread

4. Create an asynchronous class named ReadFile that you can use to read and display the image file that's stored on the device.
5. Move all code within the readFile method to the doInBackground method of the ReadFile class. Then, delete the readFile class and clean up the code so there are no syntax errors.
6. Run the app. You should get an error that indicates that you can't update the UI thread from the background thread.
7. Modify the declaration of the ReadFile class so it returns a Drawable object.
8. Modify the doInBackground method so it returns a Drawable object.
9. Modify the onPostExecute method so it uses the Drawable object to display the image.
10. Run the app. It should work as before. However, the app now uses a separate thread to read the image file.

Exercise 10-5 Use a timer thread

In this exercise, you'll modify an app that's designed to test reading and writing files.

Review and test the app

1. Start Android Studio and open the project named ch10_ex4_TestTimer.
2. Review the code for this app. Note that it uses a timer and a timer task to update the user interface every second.
3. Run the app. When it starts, it should display the number of seconds since the app started.

Modify the timer

4. Add an onPause method that cancels the timer when the app is paused.
5. Add Start and Stop buttons that allow the user to start and stop the timer.
6. Modify the timer so it only executes the timer task every 10 seconds.

Modify the timer task and update the UI

7. Add code to the timer task that downloads the RSS feed described in chapter 10. Since the timer task runs in its own thread, you don't need to use an AsyncTask class to do this.
8. Modify the code that updates the user interface so it displays a message that indicates the number of times that the app has downloaded the RSS feed. This message should be formatted like this:

```
File downloaded 3 time(s)
```

Exercise 11-3 Review the Stopwatch app and add a notification

In this exercise, you'll review a Stopwatch app. Then, you'll add a notification to it.

1. Start Android Studio and open the project named ch11_ex3_Stopwatch.
2. Run the Stopwatch app. Then, click the Start button to begin timing.
3. Navigate away from the Stopwatch app to another app such as the Home screen. Then, navigate back to the Stopwatch app. The timer should still be running.
4. Click the Stop button to stop the timer. Then, click the Reset button to reset the timer.
5. Open the class for the Stopwatch activity. Review this code until you have a general idea of how it works. Note that it saves the milliseconds for the start time in the onPause method and gets them in the onResume method. As a result, it isn't necessary to use a service for this timer.
6. Declare the NotificationManager object and its unique ID as instance variables. Then, at the end of the onCreate method, add code that creates the NotificationManager object.
7. At the end of the start method, add code that displays a notification that indicates that the stopwatch is on. If the user selects this notification from the drawer, Android should display the Stopwatch app.
8. At the end of the stop method, add code that cancels the notification. This should remove the notification from the notification area.
9. Run the Stopwatch app. Then, click the Start button to begin timing. This should display the notification.
10. Navigate away from the Stopwatch app to another app. Open the notification drawer and select the notification for the Stopwatch app. This should navigate back to the Stopwatch app, and the timer should still be running.
11. Click the Stop button to stop the timer. This should remove the notification from the notification area.

Exercise 11-4 Create a Reminder app

In this exercise, you'll add a service to a Reminder app that displays a notification every hour that says, "Look into the distance. It's good for your eyes."

Test the app

1. Start Android Studio and open the project named ch11_ex4_Reminder.
2. Review the code. Note that it contains a layout and a class for an activity, but no class for a service.
3. Run the app. Note that it displays a message that describes the app as well as a Start Service button and a Stop Service button. Note also that clicking on these buttons doesn't do anything.

Add a service

4. Add a class for a service. If this service is started or stopped, it should display an appropriate message in the LogCat view.
5. Open the Android manifest file and register the service.
6. Open the class for the activity. Then, add code that's executed when the user clicks the Start Service button. This code should start the service and display a toast that says "Service started".
7. Add code that's executed when the user clicks the Stop Service button. This code should stop the service and display a toast that says "Service stopped".
8. Run the app and test the buttons. They should display an appropriate toast on the user interface and an appropriate message in the LogCat view.

Add a timer

9. Open the service class and add code that uses a timer to print a message to the LogCat view. This message should say, "Look into the distance. It's good for your eyes!" To make this timer easy to test, you can have it execute every 10 seconds or so.
10. Run the app and check the LogCat view to make sure the timer is working correctly.

Add a notification

11. Open the service class and add code to the timer task that displays a notification that says, "Look into the distance. It's good for your eyes."
12. Run the app and make sure the notification is working correctly.
13. If necessary, modify the code for the timer so it only displays the notification every hour.

Exercise 12-2 Add broadcast receivers

In this exercise, you'll add some broadcast receivers to the Tip Calculator app. Although these broadcast receivers don't do anything useful, this exercise provides an easy way to gain some hands-on experience with adding broadcast receivers.

Add a receiver for the boot completed broadcast

1. Start Android Studio and open the project named ch12_ex2_TipCalculator.
2. Add a class for a broadcast receiver that's executed when the device finishes booting. This receiver should send a message to the LogCat view that indicates that the Tip Calculator app has received the boot completed broadcast.
3. Use the AndroidManifest.xml file to register this receiver and to add all permissions required by this receiver.
4. Test this receiver to make sure it's working correctly. To do that, run the app on a device. Then, reboot the device. When the boot completes, you should be able to view your message in the LogCat view.
5. Add code to the receiver that starts the Tip Calculator activity. To get this to work correctly, you need to add a statement that sets a flag within the intent for the activity like this:

   ```
   activity.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
   ```

6. Test this receiver to make sure it's working correctly. To do that, run the app on a device. Then, reboot the device. When the boot completes, the device should display the Tip Calculator activity.

Add a receiver for the connectivity changed broadcast

7. Add a class for a broadcast receiver that's executed when the connectivity for the device changes. This receiver should use a toast to display a message that says "Connected" or "Not Connected", depending on whether the device is connected to a network.
8. Use the AndroidManifest.xml file to register this broadcast receiver and to request all permissions required by this receiver.
9. Run the app. If you're using an actual device, test the Connectivity receiver by turning on airplane mode. This should display a toast that says:

   ```
   Not connected
   ```

   NOTE: Due to a bug in most emulators, this might not work correctly on an emulator.

10. Turn off airplane mode. This should display a toast that says:

    ```
    Connected
    ```

    NOTE: Again, this might not work correctly on an emulator.

Exercise 13-4 Add a database to the Tip Calculator app

In this exercise, you'll create a database that you can use with the Tip Calculator app.

Add a database class

1. Start Android Studio and open the project named ch13_ex4_TipCalculator.
2. Review the code. Note that it includes a Tip class that you can use to store the data for a tip.
3. Add a database class that creates a table with these column names and data types:

```
_id              INTEGER
bill_date        INTEGER
bill_amount      REAL
tip_percent      REAL
```

   When the database class creates the database, it should also insert two rows of test data. (You can use 0 for the bill date values, but make up some bill amount and tip percent values such as 40.60 and .15.)

4. Add a public getTips method that returns an ArrayList<Tip> object that contains all columns and rows from the database table.
5. Switch to the activity class. Then, add code to its onResume method that calls the getTips method and loops through all saved tips. For each saved tip, this code should use LogCat logging to send the bill date milliseconds, the bill amount, and the tip percent to the LogCat view.
6. Run the app. At this point, it should create the database, add two rows of test data, and print the data for each row to the LogCat view.

Save tip calculations

7. Switch to the database class and add a method that saves tip calculations.
8. In the layout for the activity, add a Save button below the other widgets.
9. In the class for the activity, add code that saves the current bill amount and tip percent to the database when the user clicks the Save button. This code should also clear the bill amount from the user interface.
10. Run the app and test the Save button. To do that, you can execute the onResume method by navigating away from the app and navigating back to it. Then, you can check the LogCat view to make sure the data is saved when you click on the Save button.

Display the date and time of the last saved tip

11. Switch to the database class and add a method that returns a Tip object for the last tip that was saved.
12. Add code to the onResume method that displays the date and time of the last saved tip in the LogCat view. This date and time should be in this format:

    ```
    Sep 4 2016 15:54:00
    ```

    You can use a method of the Tip class to return this format.
13. Execute the onResume method again. Then, check the LogCat view to make sure the data is displayed correctly.

Set default tip percent to average tip percent

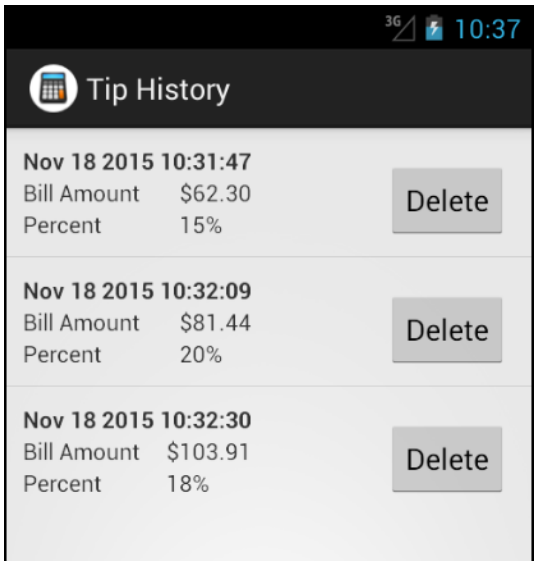
14. Switch to the database class and add a method that gets the average tip percent. To do that, you can provide an array of strings for the columns parameter like this:

    ```
    String columns[] = {"AVG(" + TIP_PERCENT + ")"};
    ```

    (This assumes the TIP_PERCENT constant contains the name of the column that stores the tip percent.)
15. Add code to the onResume method that displays the average tip percent in the LogCat view.
16. Execute the onResume method again by navigating away from the app and navigating back to it. Then, check the LogCat view to make sure the average tip percent is displayed correctly.
17. Modify the code for the Save button so it sets the tip percent to the average tip percent. That way, after you click the Save button, it should set the tip percent to the average tip percent.

Exercise 14-3 Display the data for the Tip Calculator

In this exercise, you'll display data that's stored in a database for the Tip Calculator app. To do that, you can add a Tip History activity. When you're done, it should look like this:



Test the app

1. Start Android Studio and open the project named ch14_ex3_TipCalculator. This app uses a database class to store tips in a database.
2. Review the code for the database class and the Tip Calculator activity class. Note that it includes a database class (TipDB) and a business class (Tip). If you did exercise 13-4, most of this code should be familiar to you.
3. Run the app and use it to calculate and save a tip. Then, click on the View History button. Note that this button doesn't do anything yet.

Add the Tip History activity

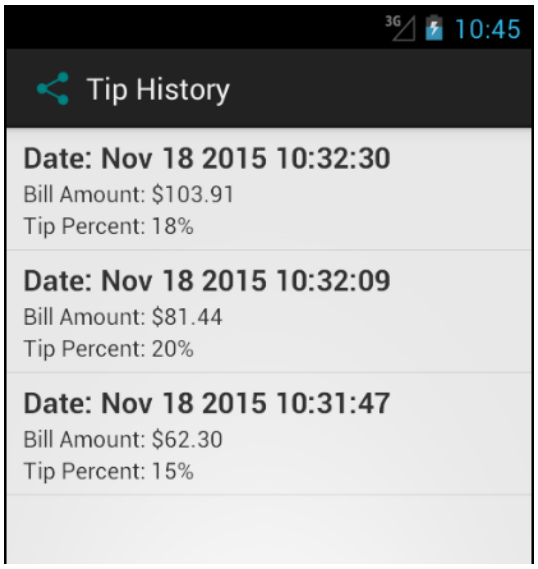
4. Add the layout and class for the Tip History activity. The layout should be a vertical linear layout that includes a single ListView widget.
5. Add code that displays the Tip History activity when the user clicks on the View History button.
6. Run the app and click on the View History button. This should display a blank screen.

Add the custom adapter

7. Add the layout for each item in the ListView widget. This should include the date, bill amount, tip percent, and a Delete button.
8. Add a class that extends the layout for each item in the ListView widget. This class should display the correct data on the layout.
9. In the class that extends the layout, add an event handler for the Delete button. If the user clicks this button, this event handler should delete the saved tip from the database. (There's a method in the database class that you can use to do this.) In addition, it should refresh the user interface. (You can start the Tip History activity again to do that.)
10. Create a custom adapter for the ListView widget.
11. Modify the onResume method for the Tip History activity so it gets the data from the database, creates the adapter, and sets the adapter for the ListView widget.
12. Run the app. Now, clicking the View History button should display the saved tasks. In addition, you should be able to delete a saved tip by clicking on its Delete button.

Exercise 15-3 Add a content provider to the Tip Calculator and use it

In this exercise, you'll add a content provider to a Tip Calculator app that saves its tips in a database. Then, you'll create a Tip History app that uses the same database. When you're done, the Tip History app should look like this:



Test the app

1. Start Android Studio and open the project named ch15_ex3_TipCalculator. This app uses a database class to store tips in a database.
2. Review the code for the database class and the Tip Calculator activity class. Note that it includes a database class (TipDB) and a business class (Tip). If you did exercise 13-4 or 14-3, most of this code should be familiar to you.
3. Run the app and use it to calculate and save a tip.

Add a content provider

4. Open the TipDB class and add supporting methods that provide for querying, inserting, and deleting rows, but not for updating rows.
5. Add a content provider class to the project and code it so it works like the content provider class presented in this chapter. However, the update method should always throw an exception that indicates the operation is not supported.
6. Open the AndroidManifest.xml file, and register the content provider.
7. Run the app. If the app compiles and you don't get any error messages, the content provider is available to other apps.

Create a Task History app that uses the content provider

8. Create a new project for a Tip History app. If you want, you can do that by creating a copy of the Task History app presented in chapter 15.
9. Copy the necessary database and content provider constants from the TipDB class you modified earlier in this exercise to the Tip History activity that you just created.
10. Modify the layouts for the Tip History activity so they use a ListView widget to diplay the saved tips.
11. Modify the class for the Tip History activity so it displays the saved tips in the ListView widget.
12. Add an event handler that handles the click event for an item in the ListView widget. This event handler should display a dialog box that allows the user to confirm or cancel the deletion of the item that was clicked.
13. If necessary, modify the AndroidManifest.xml file so it refers to the correct package and activity.
14. Run the app. You should be able to view all saved tips on the device, and you should be able to delete them by clicking on them.

Exercise 16-3 Add an app widget to the News Reader app

In this exercise, you'll add an app widget to the News Reader app presented in chapter 12. When you're done, this app widget should look something like this:

1. Start Android Studio and open the project named ch16_ex2_NewsReader.
2. Open the FileIO class and note that it includes a method named readFile that reads the current RSS feed from a file.
3. Create a layout named app_widget_top_headlines that uses four TextView widgets to display the name of the app and the titles for the three most recent headlines.
4. Create a class named AppWidgetTopHeadlines that displays the three most recent headlines. To do that, you can use a FileIO object to read the RSS feed from a file. This class should update the data on the layout when the widget is first added to the Home screen and when the NEW_FEED action is received. (This action is stored in the RSSFeed class.)
5. Create an info file for the app widget that specifies that the icon is 4 cells wide by 1 cell tall. This info file should not request an update.
6. Open the Android manifest file and add a receiver element that registers the app widget.
7. Run the app on an emulator or device. Note the top headlines that are displayed by the app.
8. Add the app widget to the Home screen and check the top headlines to make sure the app widget is working correctly. If one of the first two headlines is too long to fit on a single line, the app widget may not have enough space to display all three headlines (as shown above).

Exercise 17-3 Install the Task List app on your device

In this exercise, you'll install the release build of the Tip Calculator app on an Android device.

Create a signed APK file

1. Start Android Studio and open the project named ch17_ex3_TaskList.
2. Create the signed APK file and store it in the ch17_ex3_TaskList directory. When you do this, you should also create a new key store file that stores a new release key.

Side load the APK file onto a device

3. Select the Android device that you want to use.
4. Uninstall any old versions of the Task List app that use the debug key.
5. Make sure the Android device allows apps from sources other than Google Play. To do that, you can use the Settings app.
6. Copy the APK file that's in the ch17_ex3_TaskList directory to the device. One easy way to do this is to attach it to an email and send it to a Gmail account. Then, you can open Gmail on the device, open the email, and click on the attachment.
7. Click the APK file to install the Task List app.
8. Start the Task List app and make sure it works correctly.

Exercise 18-2 Modify the Location Viewer app

In this exercise, you'll start by testing a simple app that displays the latitude and longitude for the current location of a device. Then, you'll enhance this app so it updates the current location every 10 seconds and displays that location on a map.

This exercise only describes how to test on an actual device, not on an emulator. In addition, this device must have a valid version of Google Play installed, and it must support OpenGL ES version 2.

Test the Location Viewer app

1. Start Android Studio and open the project named ch18_ex3_LocationViewer.
2. Open the Project Structure dialog and make sure that the play-services library is a dependency. If it's not, add it.
3. Review the code. To start, open the layout and class for the activity and note that it just displays the current latitude and longitude. Then, open the Android manifest file and note that it only requests one permission: the ACCESS_FINE_LOCATION permission.
4. Connect the device you want to use for testing to your computer.
5. Run the Location Viewer app on this device. This should display the latitude and longitude for the current location of the device. If you change locations, this app does not update the latitude and longitude.

Add location updates

6. Open the class for the activity and implement the LocationListener interface so that the Location Viewer app updates the location at least every 10 seconds but can update the location every second if necessary.
7. Run the app. This should display the latitude and longitude for the current location of the device. If you change locations, this app should update the latitude and longitude.

Display a map

8. Open the layout for the activity and add a fragment for a map that's displayed below the TextView widget.
9. Open the class for the activity and add code that displays a map with all of its default settings.
10. Get an API key for Google Maps.
11. Open the manifest file for the app and add the permissions you need to display a map. In addition, add a meta-data element that specifies the name and value for the API key. In other words, store the API key directly in the Manifest file, not as a string resource in a separate file.
12. Run the app on the device. This should display a map. If it doesn't, check the LogCat view for error messages and troubleshoot the problem. To refresh the API key, you may need to uninstall the app before running it again.

Zoom in and display a marker

13. Open the class for the activity and add code that zooms in on the current location. You can choose settings you like for the zoom level, bearing, and tilt.
14. Run the app. It should zoom in on the current location, but it shouldn't display a marker for the current location.
15. Open the class for the activity and add code that clears any other markers from the map and then displays a marker for the current location.
16. Run the app. It should zoom in on the current location and display a single marker for that location.
17. Go outside and walk for a while. The map should move the marker to reflect your current location (or at least somewhat close to your current location).