15.6 `FileChooser` and `DirectoryChooser` Dialogs

JavaFX classes `FileChooser` and `DirectoryChooser` (package `javafx.stage`) display dialogs that enable the user to select a file or directory, respectively. To demonstrate these dialogs, we enhance the example in Section 15.3. The example (Figs. 15.14–15.15) contains a JavaFX graphical user interface, but still displays the same data as the earlier example.

Creating the JavaFX GUI

The GUI (Fig. 15.15(a)) consists of a 600-by-400 `BorderPane` with the **fx:id** `borderPane`:

- In the `BorderPane`'s top, we placed a `ToolBar` layout (from the Scene Builder **Library**'s **Containers** section), which arranges its controls horizontally (by default) or vertically. Typically, you place `ToolBar`s at your GUI's edges, such as in a `BorderPane`'s top, right, bottom or left areas.
- In the `BorderPane`'s center, we placed a `TextArea` control with the **fx:id** `textArea`. We set the control's **Text** property to `"Select file or directory"` and enabled its **Wrap Text** property to ensure that long lines of text wrap to the next line. If there are more lines of text to display than vertical lines in the `TextArea`, the control will show a vertical

scrollbar. (When **Wrap Text** is not enabled, the `TextArea` also shows a horizontal scrollbar if the text is too wide to display.)

By default, the `ToolBar` you drag onto your layout has one `Button`. You can drag other controls onto the `ToolBar` and, if necessary, remove the default `Button`. We added a second `Button`. For the first `Button`, we set:

- the **Text** property to `"Select File"`,
- the **fx:id** property to `selectFileButton` and
- the **On Action** event handler to `selectFileButtonPressed`.

For the second `Button`, we set:

- the **Text** property to `"Select Directory"`,
- the **fx:id** property to `selectDirectoryButton` and
- the **On Action** event handler to `selectDirectoryButtonPressed`.

Finally, we specified `FileChooserTestController` as the FXML's controller.

Class That Launches the App

Class `FileChooserTest` (Fig. 15.14) launches the JavaFX application, using the same techniques you learned in Chapters 12–13.

```
33    // Fig. 15.14: FileChooserTest.java
34    // App to test classes FileChooser and Director
35    import javafx.application.Application;
36    import javafx.fxml.FXMLLoader;
37    import javafx.scene.Parent;
38    import javafx.scene.Scene;
39    import javafx.stage.Stage;
40
41    public class FileChooserTest extends Applicatio
42       @Override
43       public void start(Stage stage) throws Except
44          Parent root =
45             FXMLLoader.load(getClass().getResource
46
47          Scene scene = new Scene(root);
48          stage.setTitle("File Chooser Test"); // d
49          stage.setScene(scene);
50          stage.show();
51       }
52
53       public static void main(String[] args) {
54          launch(args);
55       }
56    }
```

Fig. 15.14

Demonstrating `JFileChooser`.

Controller Class

Class `FileChooserTestController` (Fig. 15.15)

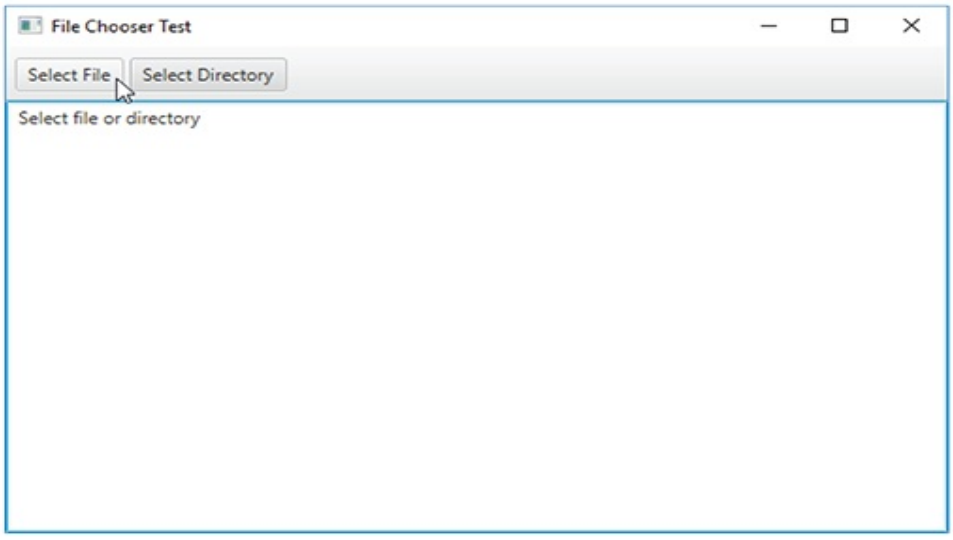
responds to the `Button`s' events. Both event handlers call method `analyzePath` (defined in lines 70–110) to determine whether a `Path` is a file or directory, display information about the `Path` and, if it's a directory, list its contents.

```
1    // Fig. 15.15: FileChooserTestController.java
2    // Displays information about a selected file o
3    import java.io.File;
4    import java.io.IOException;
5    import java.nio.file.DirectoryStream;
6    import java.nio.file.Files;
7    import java.nio.file.Path;
8    import java.nio.file.Paths;
9    import javafx.event.ActionEvent;
10   import javafx.fxml.FXML;
11   import javafx.scene.control.Button;
12   import javafx.scene.control.TextArea;
13   import javafx.scene.layout.BorderPane;
14   import javafx.stage.DirectoryChooser;
15   import javafx.stage.FileChooser;
16
17   public class FileChooserTestController {
18      @FXML private BorderPane borderPane;
19      @FXML private Button selectFileButton;
20      @FXML private Button selectDirectoryButton;
21      @FXML private TextArea textArea;
22
23      // handles selectFileButton's events
24      @FXML
25      private void selectFileButtonPressed(ActionE
26         // configure dialog allowing selection of
27         FileChooser fileChooser = new FileChooser
28         fileChooser.setTitle("Select File");
29
30         // display files in folder from which the
31         fileChooser.setInitialDirectory(new File(
```

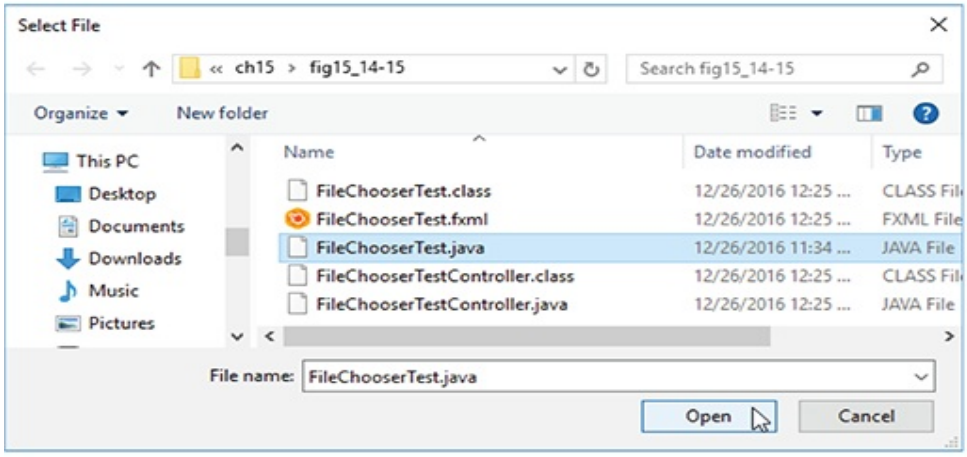
```
                    32
      33          // display the FileChooser
34          File file = fileChooser.showOpenDialog(
35             borderPane.getScene().getWindow());
                    36
37          // process selected Path or display a mes
         38          if (file != null) {
   39             analyzePath(file.toPath());
                    40          }
                 41          else {
42             textArea.setText("Select file or direc
                    43          }
                      44       }
                          45
46       // handles selectDirectoryButton's events
                   47       @FXML
48       private void selectDirectoryButtonPressed(Ac
49          // configure dialog allowing selection of
50          DirectoryChooser directoryChooser = new D
51          directoryChooser.setTitle("Select Directo
                    52
53          // display folder from which the app was
54          directoryChooser.setInitialDirectory(new
                    55
     56          // display the FileChooser
57          File file = directoryChooser.showDialog(
58             borderPane.getScene().getWindow());
                    59
60          // process selected Path or display a mes
         61          if (file != null) {
    62             analyzePath(file.toPath());
                    63          }
                 64          else {
65             textArea.setText("Select file or direc
                    66          }
                      67       }
                          68
69       // display information about file or directo
  70       public void analyzePath(Path path) {
                 71          try {
```

```
72             // if the file or directory exists, di
73             if (path != null && Files.exists(path)
74                // gather file (or directory) infor
75                StringBuilder builder = new StringB
76                builder.append(String.format("%s:%n
77                builder.append(String.format("%s a
78                   Files.isDirectory(path) ? "Is" :
79                builder.append(String.format("%s an
80                   path.isAbsolute() ? "Is" : "Is n
81                builder.append(String.format("Last
82                   Files.getLastModifiedTime(path))
83                builder.append(String.format("Size:
84                builder.append(String.format("Path:
85                builder.append(String.format("Absol
  86                    path.toAbsolutePath()));
                       87
88                if (Files.isDirectory(path)) { // o
89                   builder.append(String.format("%n
                        90
91                   // object for iterating through
92                   DirectoryStream<Path> directoryS
93                      Files.newDirectoryStream(path
                      94
95                   for (Path p : directoryStream) {
96                      builder.append(String.format(
            97                     }
             98                 }
                      99
100             // display file or directory info
101             textArea.setText(builder.toString()
               102             }
  103           else { // Path does not exist
104             textArea.setText("Path does not exi
               105             }
                106          }
  107         catch (IOException ioException) {
108             textArea.setText(ioException.toString(
                109          }
                  110      }
                    111   }
```

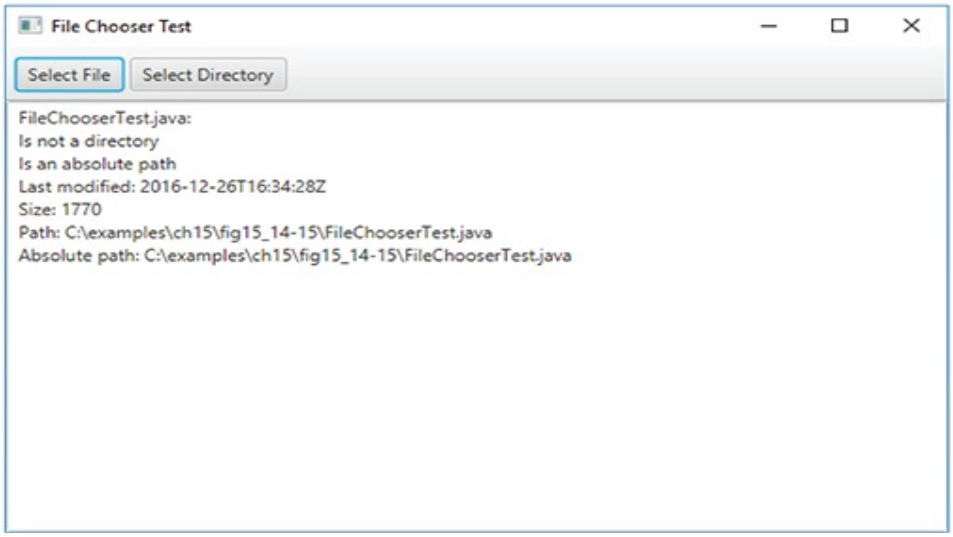
a) Initial app window.



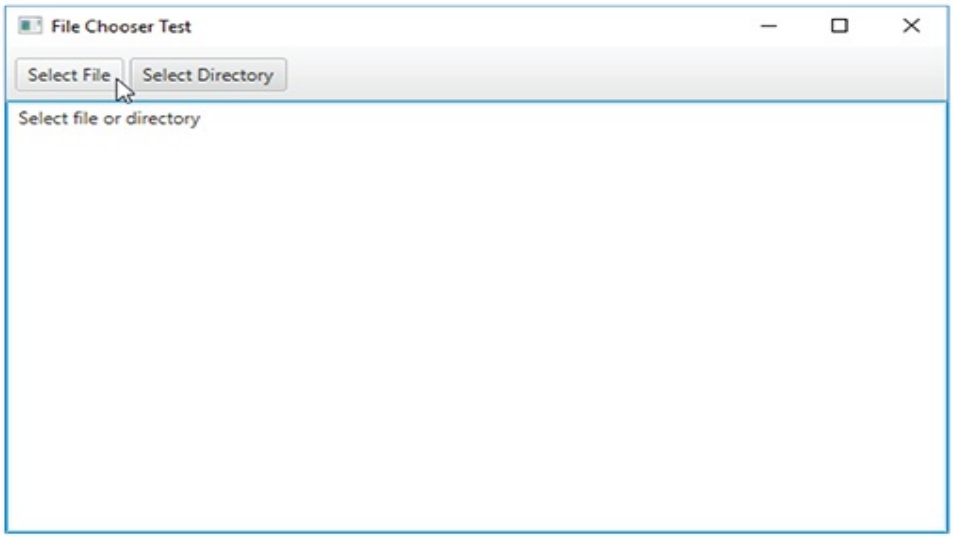
b) Selecting `FileChooserTest.java` from the `FileChooser` dialog displayed when the user clicked the **Select File** `Button`.



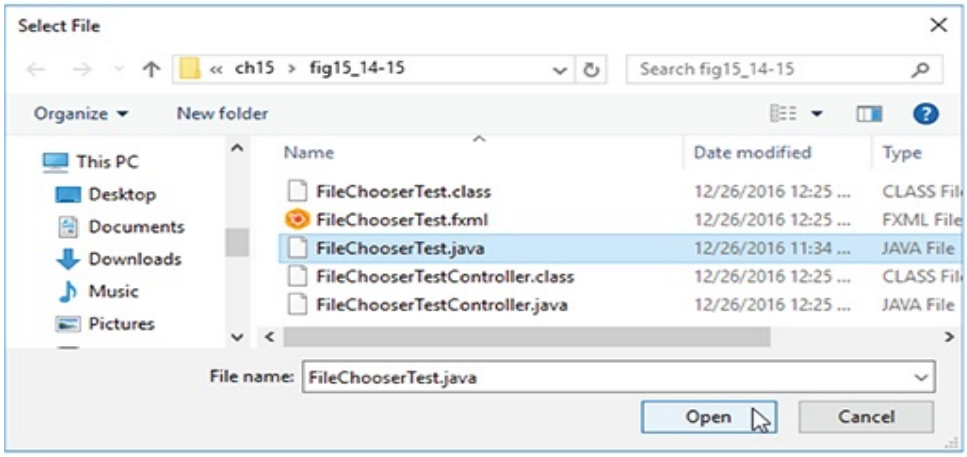
c) Displaying information about the file `FileChooserTest.java`.



a) Initial app window.



b) Selecting `FileChooserTest.java` from the `FileChooser` dialog displayed when the user clicked the **Select File** `Button`.



c) Displaying information about the file `FileChooserTest.java`.

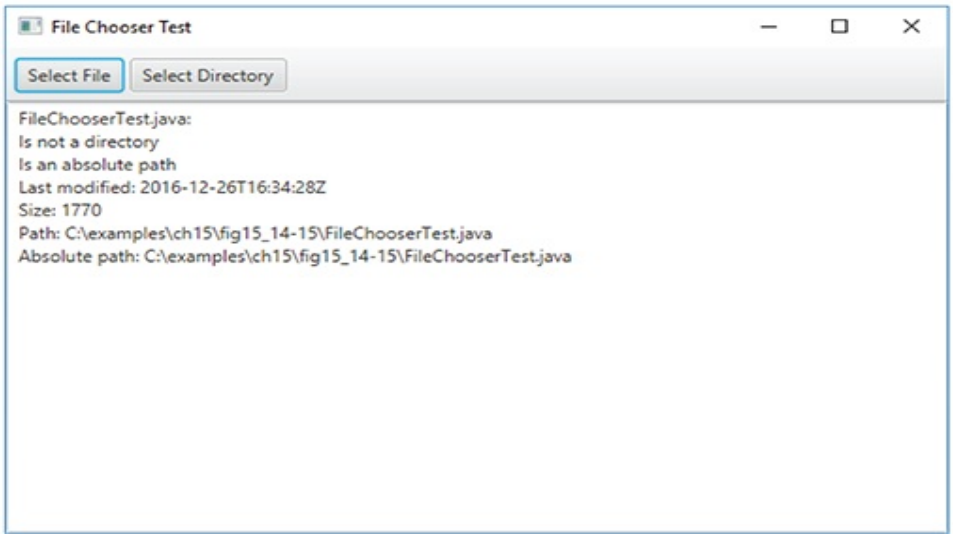


Fig. 15.15

Displays information about a selected file or folder.

Description

Method `selectFileButtonPressed`

When the user presses the **Select File** button, method `selectFileButtonPressed` (lines 24–44) creates, configures and displays a `FileChooser`. Line 28 sets the text displayed in the `FileChooser`'s title bar. Line 31 specifies the initial directory that should be opened when the `FileChooser` is displayed. Method `setInitialDirectory` receives a `File` object representing the directory's location—`"."` represents the current folder from which the app was launched.

Lines 34–35 display the `FileChooser` by calling its `showOpenDialog` method to display a dialog with an **Open** button for opening a file. There's also a `showSaveDialog` method that displays a dialog with a **Save** button for saving a file. This method receives as its argument a reference to the app's `Window`. A non-`null` argument makes the

`FileChooser` a modal dialog that prevents the user from interacting with the rest of the app until the dialog is dismissed—when the user selects a file or clicks **Cancel**. To obtain the app's `Window`, we use the `borderPane`'s `getScene` method to get a reference to its parent `Scene`, then use the `Scene`'s `getWindow` method to get a reference to the `Window` containing the `Scene`.

Method `showOpenDialog` returns a `File` representing the selected file's location, or `null` if the user clicks the **Cancel** button. If the `File` is not `null`, line 39 calls `analyzePath` to display the selected file's information—`File` method `toPath` returns a `Path` object representing the location. Otherwise, line 42 displays a message in the `TextArea` telling the user to select a file or directory. The screen captures in Fig. 15.15(b) and (c) show the `FileChooser` dialog with the `FileChooserTest.java` file selected and, after the user presses the **Open** button, the file's information displayed.

Method `selectDirectoryButtonPressed`

When the user presses the **Select Directory** button, method `selectDirectoryButtonPressed` (lines 47–67) creates, configures and displays a `DirectoryChooser`. The method performs the same tasks as method `selectFileButtonPressed`. The key difference is line

57, which calls `DirectoryChooser` method `showDialog` to display the dialog—there are not separate open and save dialogs for selecting folders. Method `showDialog` returns a `File` representing the location of the selected directory, or `null` if the user clicks **Cancel**. If the `File` is not `null`, line 62 calls `analyzePath` to display information about the selected directory. Otherwise, line 65 displays a message in the `TextArea` telling the user to select a file or directory. The screen captures in Fig. 15.15(d) and (e) show the `FileChooser` dialog with the `fig15_14-15` directory selected and, after the user presses the **Open** button, the directory's information displayed.