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.

```java
// Fig. 15.14: FileChooserTest.java
// App to test classes FileChooser and Director
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class FileChooserTest extends Applicatio
   @Override
   public void start(Stage stage) throws Except
      Parent root =
         FXMLLoader.load(getClass().getResource

      Scene scene = new Scene(root);
      stage.setTitle("File Chooser Test"); // d
      stage.setScene(scene);
      stage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

Fig. 15.14

Demonstrating `JFileChooser`.

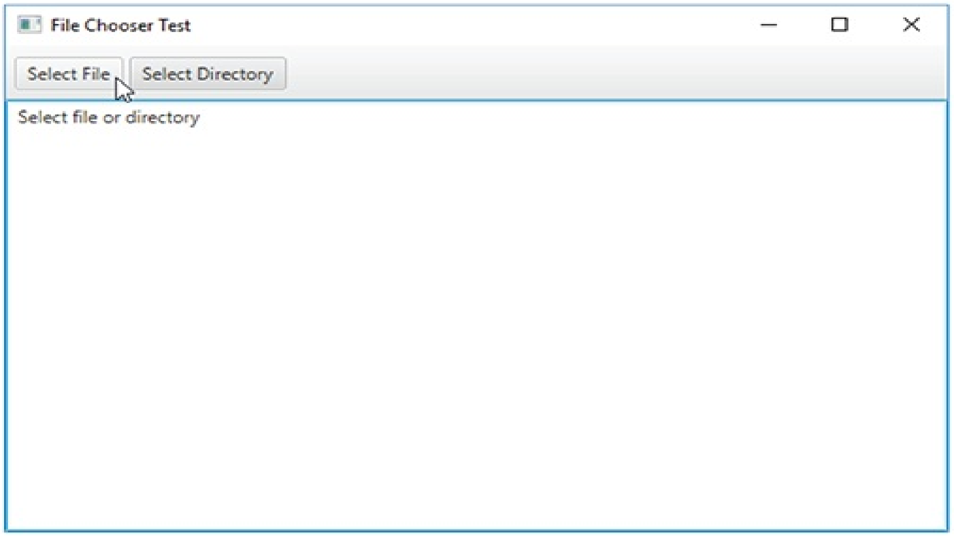
Controller Class

Class `FileChooserTestController` (Fig. 15.15) responds to the `Buttons`’ 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.

```java
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(

        // display the FileChooser
        File file = fileChooser.showOpenDialog(
            borderPane.getScene().getWindow());

        // process selected Path or display a mes
        if (file != null) {
            analyzePath(file.toPath());
        }
        else {
            textArea.setText("Select file or direc
        }
    }

    // handles selectDirectoryButton's events
    @FXML
    private void selectDirectoryButtonPressed(Ac
        // configure dialog allowing selection of
        DirectoryChooser directoryChooser = new D
        directoryChooser.setTitle("Select Directo

        // display folder from which the app was
        directoryChooser.setInitialDirectory(new

        // display the FileChooser
        File file = directoryChooser.showDialog(
            borderPane.getScene().getWindow());

        // process selected Path or display a mes
        if (file != null) {
            analyzePath(file.toPath());
        }
        else {
            textArea.setText("Select file or direc
        }
    }

    // display information about file or directo
    public void analyzePath(Path path) {
        try {
```

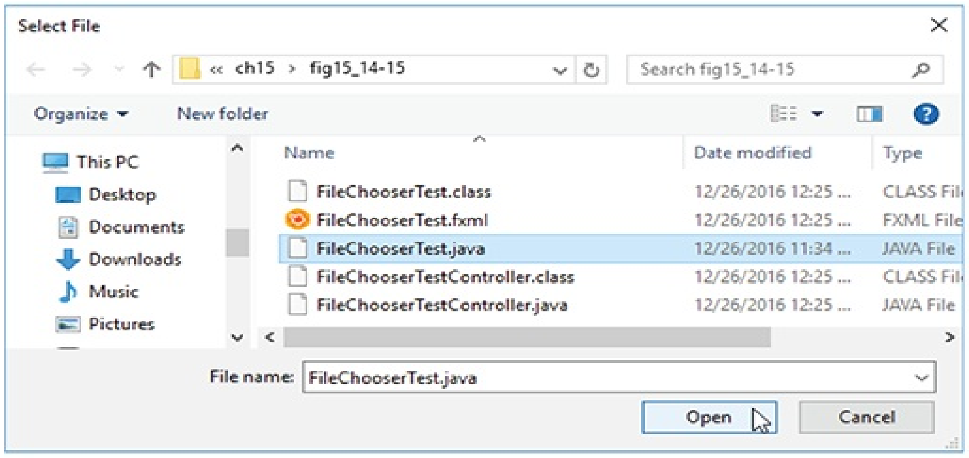
```
      // if the file or directory exists, di
      if (path != null && Files.exists(path)
        // gather file (or directory) infor
        StringBuilder builder = new StringB
        builder.append(String.format("%s:%n
        builder.append(String.format("%s a
          Files.isDirectory(path) ? "Is" :
        builder.append(String.format("%s an
          path.isAbsolute() ? "Is" : "Is n
        builder.append(String.format("Last
          Files.getLastModifiedTime(path))
        builder.append(String.format("Size:
        builder.append(String.format("Path:
        builder.append(String.format("Absol
          path.toAbsolutePath()));

        if (Files.isDirectory(path)) { // o
          builder.append(String.format("%n

          // object for iterating through
          DirectoryStream<Path> directoryS
            Files.newDirectoryStream(path

          for (Path p : directoryStream) {
            builder.append(String.format(
          }
        }

        // display file or directory info
        textArea.setText(builder.toString()
      }
      else { // Path does not exist
        textArea.setText("Path does not exi
      }
    }
    catch (IOException ioException) {
      textArea.setText(ioException.toString(
    }
  }
}
```

a) Initial app window.



Select file or directory

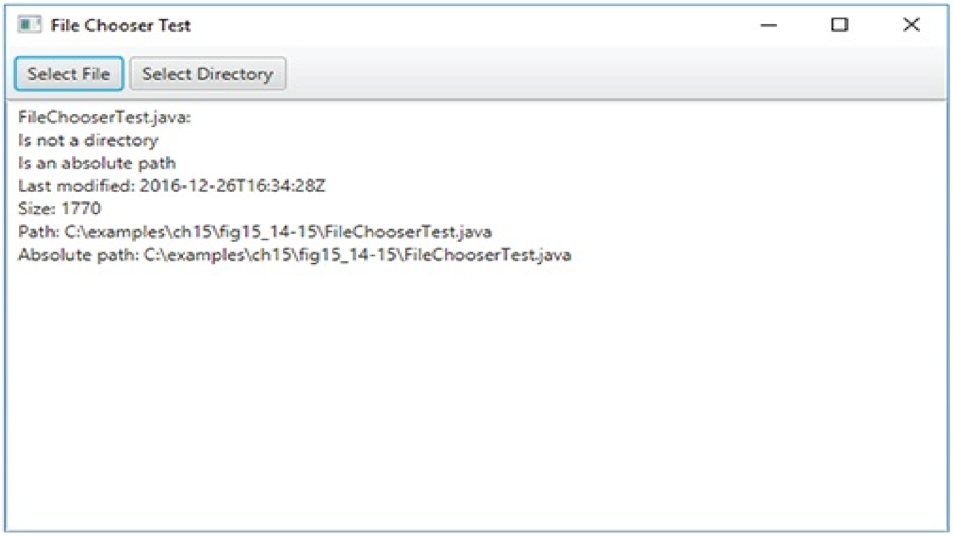
b) Selecting `FileChooserTest.java` from the `FileChooser` dialog displayed when the user clicked the **Select File** `Button`.



Name	Date modified	Type
FileChooserTest.class	12/26/2016 12:25 ...	CLASS Fil
FileChooserTest.fxml	12/26/2016 12:25 ...	FXML File
FileChooserTest.java	12/26/2016 11:34 ...	JAVA File
FileChooserTestController.class	12/26/2016 12:25 ...	CLASS Fil
FileChooserTestController.java	12/26/2016 12:25 ...	JAVA File

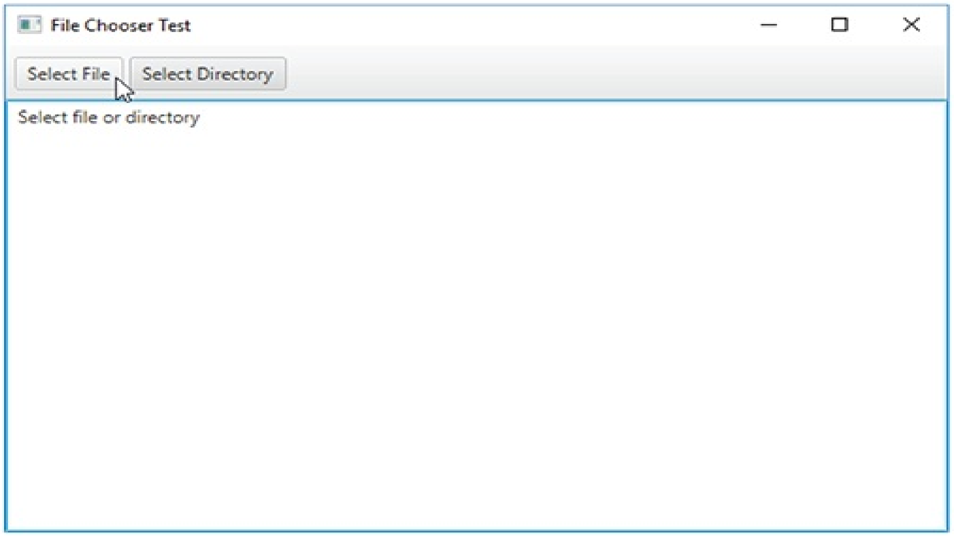
File name: FileChooserTest.java

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



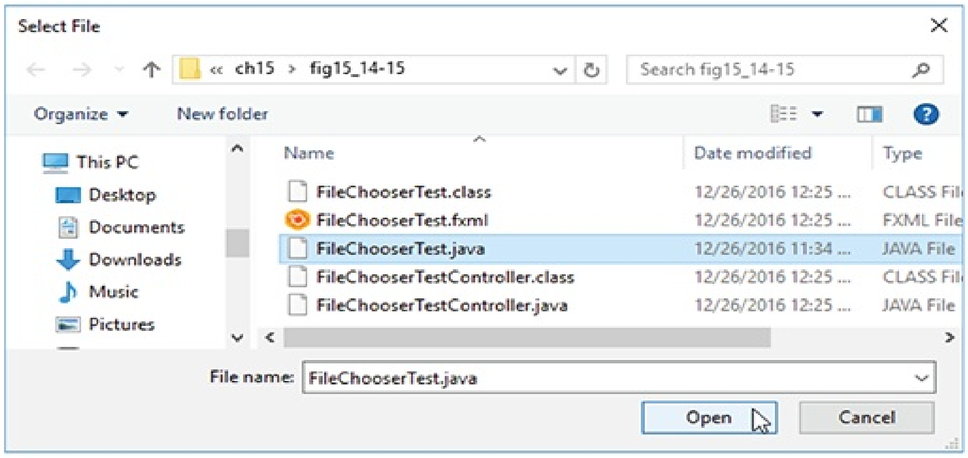
```
FileChooserTest.java:
Is not a directory
Is an absolute path
Last modified: 2016-12-26T16:34:28Z
Size: 1770
Path: C:\examples\ch15\fig15_14-15\FileChooserTest.java
Absolute path: C:\examples\ch15\fig15_14-15\FileChooserTest.java
```

a) Initial app window.



Select file or directory

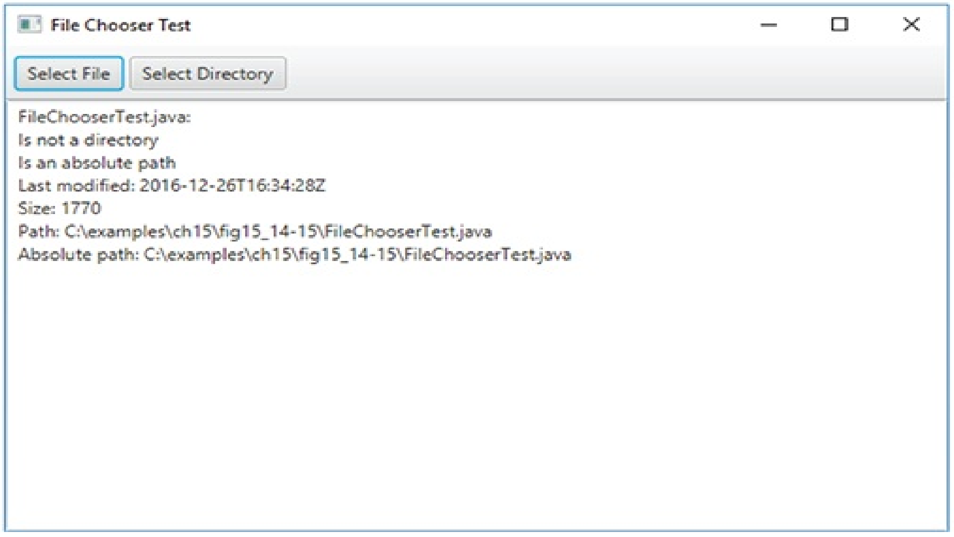
b) Selecting `FileChooserTest.java` from the `FileChooser` dialog displayed when the user clicked the **Select File** `Button`.



Name	Date modified	Type
FileChooserTest.class	12/26/2016 12:25 ...	CLASS Fil
FileChooserTest.fxml	12/26/2016 12:25 ...	FXML File
FileChooserTest.java	12/26/2016 11:34 ...	JAVA File
FileChooserTestController.class	12/26/2016 12:25 ...	CLASS Fil
FileChooserTestController.java	12/26/2016 12:25 ...	JAVA File

File name: FileChooserTest.java

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



```
FileChooserTest.java:
Is not a directory
Is an absolute path
Last modified: 2016-12-26T16:34:28Z
Size: 1770
Path: C:\examples\ch15\fig15_14-15\FileChooserTest.java
Absolute path: C:\examples\ch15\fig15_14-15\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.