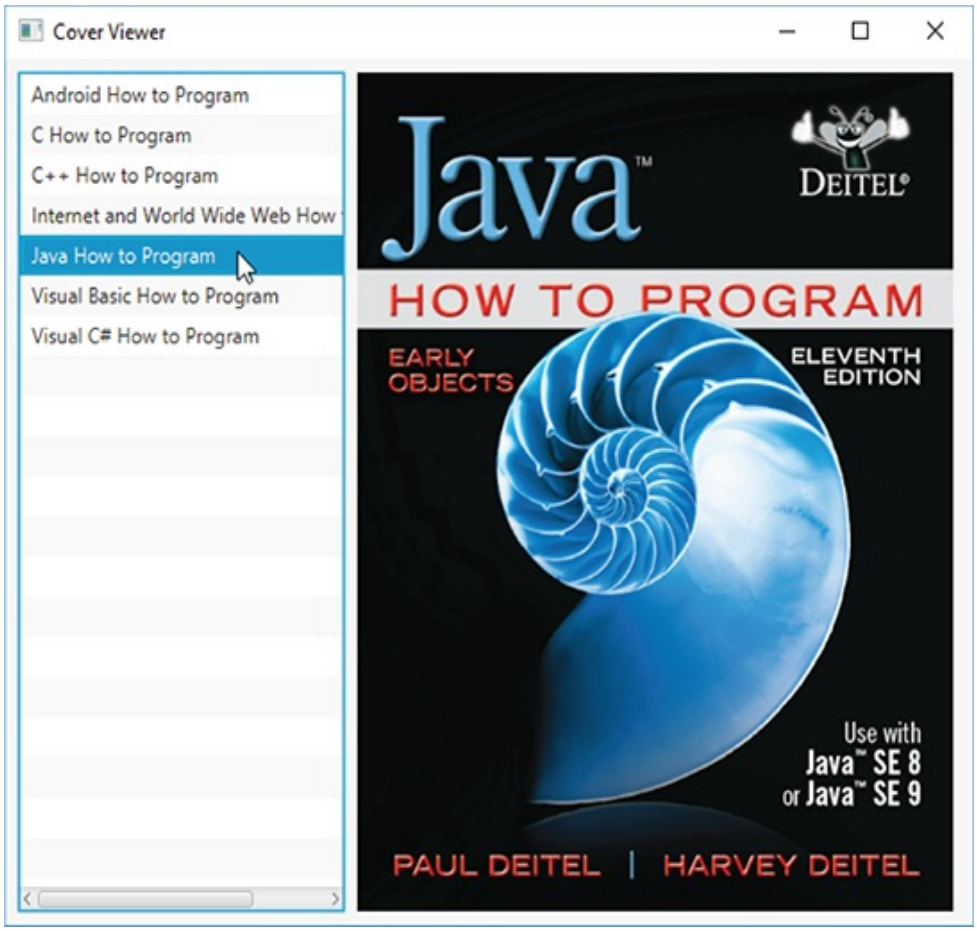
# 13.5 Cover Viewer App: Data-Driven GUIs with JavaFX Collections

Often an app needs to edit and display data. JavaFX provides a comprehensive model for allowing GUIs to interact with data. In this section, you'll build the **Cover Viewer** app (Fig. 13.12), which binds a list of `Book` objects to a `ListView`. When the user selects an item in the `ListView`, the corresponding `Book`'s cover image is displayed in an `ImageView`.




# Fig. 13.12

**Cover Viewer** with Java How to Program selected.

Description

# 13.5.1 Technologies Overview

This app uses a `ListView` control to display a collection of

book titles. Though you can individually add items to a `ListView`, in this app you'll bind an `ObservableList` object to the `ListView`. If you make changes to an `ObservableList`, its observer (the `ListView` in this app) will automatically be notified of those changes. Package `javafx.collections` defines `ObservableList` (similar to an `ArrayList`) and other observable collection interfaces. The package also contains class `FXCollections`, which provides `static` methods for creating and manipulating observable collections. You'll use a property listener to display the correct image when the user selects an item from the `ListView`—in this case, the property that changes is the selected item.

## 13.5.2 Adding Images to the App's Folder

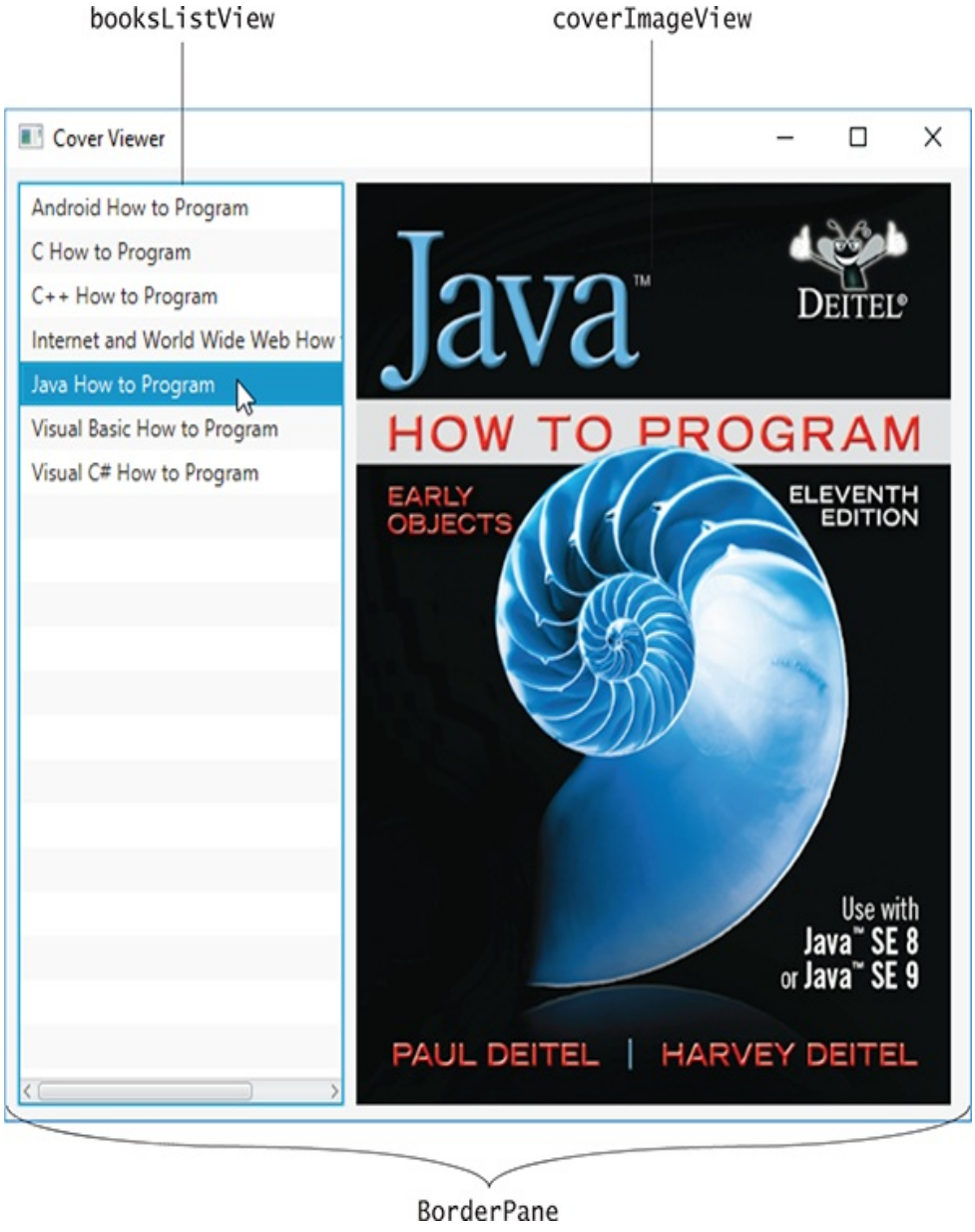
From this chapter's examples folder, copy the `images` folder (which contains the `large` and `small` subfolders) into the folder where you'll save this app's FXML file, and the source-code files `CoverViewer.java` and `CoverViewerController.java`. Though you'll use only the `large` images in this example, you'll copy this app's folder to create the next example, which uses both sets of images.

## 13.5.3 Building the GUI

In this section, we'll discuss the **Cover Viewer** app's GUI. As you've done previously, create a new FXML file, then save it as `CoverViewer.fxml`.

# fx:id Property Values for This App's Controls

Figure 13.13 shows the **fx:id** properties of the **Cover Viewer** app's programmatically manipulated controls. As you build the GUI, you should set the corresponding **fx:id** properties in the FXML document.




# Fig. 13.13

**Cover Viewer** app's programmatically manipulated controls labeled with their **fx:id**s.

# Adding and Configuring the Controls

Using the techniques you learned previously, create a `BorderPane`. In the left area, place a `ListView` control, and in the center area, place an `ImageView` control.

For the `ListView`, set the following properties:

- **Margin**—`8` (for the right margin) to separate the `ListView` from the `ImageView`
- **Pref Width**—`200`
- **Max Height**—`MAX_VALUE`
- **Min Width**, **Min Height**, **Pref Height** and **Max Width** —`USE_COMPUTED_SIZE`

For the `ImageView`, set the **Fit Width** and **Fit Height** properties to `370` and `480`, respectively. To size the `BorderPane` based on its contents, set its **Pref Width** and **Pref Height** to `USE_COMPUTED_SIZE`. Also, set the **Padding** property to `8` to inset the `BorderPane` from the stage.

# Specifying the Controller Class's Name

To ensure that an object of the controller class is created when the app loads the FXML file at runtime, specify

`CoverViewerController` as the controller class's name in the FXML file as you've done previously.

## Generating a Sample Controller Class

Select **View** > **Show Sample Controller Skeleton**, then copy this code into a `CoverViewerController.java` file and store the file in the same folder as `CoverViewer.fxml`. We show the completed `CoverViewerController` class in Section 13.5.5.

## 13.5.4 CoverViewer Subclass of `Application`

Figure 13.14 shows class `CoverViewer` subclass of `Application`.

```
   1  // Fig. 13.13: CoverViewer.java
2  // Main application class that loads and display
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  public class CoverViewer extends Application {
10     @Override
```

```
11      public void start(Stage stage) throws Excepti
            12          Parent root =
13            FXMLLoader.load(getClass().getResource(
                        14
    15          Scene scene = new Scene(root);
    16          stage.setTitle("Cover Viewer");
        17          stage.setScene(scene);
             18          stage.show();
                     19      }
                         20
 21      public static void main(String[] args) {
             22          launch(args);
                     23      }
                       24   }
```

# Fig. 13.14

Main application class that loads and displays the **Cover Viewer**'s GUI.

## 13.5.5 `CoverViewerController` Class

Figure 13.15 shows the final version of class `CoverViewerController` with the app's new features highlighted.

```
1  // Fig. 13.14: CoverViewerController.java
2  // Controller for Cover Viewer application
3  import javafx.beans.value.ChangeListener;
4  import javafx.beans.value.ObservableValue;
5  import javafx.collections.FXCollections;
6  import javafx.collections.ObservableList;
7  import javafx.fxml.FXML;
8  import javafx.scene.control.ListView;
9  import javafx.scene.image.Image;
10 import javafx.scene.image.ImageView;
11
12 public class CoverViewerController {
13    // instance variables for interacting with GU
14    @FXML private ListView<Book> booksListView;
15    @FXML private ImageView coverImageView;
16
17    // stores the list of Book Objects
18    private final ObservableList<Book>books =
19       FXCollections.observableArrayList();
20
21    // initialize controller
22    public void initialize() {
23       // populate the ObservableList<Book>
24       books.add(new Book("Android How to Program
25          "/images/small/androidhtp.jpg", "/image
26       books.add(new Book("C How to Program",
27          "/images/small/chtp.jpg", "/images/larg
28       books.add(new Book("C++ How to Program",
29          "/images/small/cpphtp.jpg", "/images/la
30       books.add(new Book("Internet and World Wid
31          "/images/small/iw3htp.jpg", "/images/la
32       books.add(new Book("Java How to Program",
33          "/images/small/jhtp.jpg", "/images/larg
34       books.add(new Book("Visual Basic How to Pr
35          "/images/small/vbhtp.jpg", "/images/lar
36       books.add(new Book("Visual C# How to Progr
37          "/images/small/vcshtp.jpg", "/images/la
38       booksListView.setItems(books); // bind boo
39
40       // when ListView selection changes, show l
```

```
41          booksListView.getSelectionModel().selected
42             addListener(
43                new ChangeListener<Book>() {
44                   @Override
45                public void changed(ObservableVal
46                   Book oldValue, Book newValue)
47                   coverImageView.setImage(
48                      new Image(newValue.getLarge
49                   }
50                }
51             );
52       }
53    }
```

Fig. 13.15

Controller for **Cover Viewer** application.

# @FXML Instance Variables

Lines 14–15 declare the controller's `@FXML` instance variables. Notice that `ListView` is a generic class. In this case, the `ListView` displays `Book` objects. Class `Book` contains three `String` instance variables with corresponding *set* and *get* methods:

- `title`—the book's title.
- `thumbImage`—the path to the book's thumbnail image (used in the next example).
- `largeImage`—the path to the book's large cover image.

The class also provides a `toString` method that returns the `Book`'s title and a constructor that initializes the three instance variables. You should copy class `Book` from this chapter's examples folder into the folder that contains `CoverViewer.fxml`, `CoverViewer.java` and `CoverViewerController.java`.

# Instance Variable `books`

Lines 18–19 define the `books` instance variable as an `ObservableList<Book>` and initialize it by calling `FXCollections static` method `observableArrayList`. This method returns an empty collection object (similar to an `ArrayList`) that implements the `ObservableList` interface.

# Initializing the `books` `ObservableList`

Lines 24–37 in method `initialize` create and add `Book` objects to the `books` collection. Line 38 passes this collection to `ListView` method `setItems`, which binds the `ListView` to the `ObservableList`. This *data binding* allows the `ListView` to display the `Book` objects automatically. By default, the `ListView` displays each `Book`'s `String` representation. (In the next example, you'll customize this.)

# Listening for `ListView` Selection Changes

To synchronize the book cover that's being displayed with the currently selected book, we listen for changes to the `ListView`'s selected item. By default a `ListView` supports single selection—one item at a time may be selected. `ListView`s also support multiple selection. The type of selection is managed by the `ListView`'s `MultipleSelectionModel` (a subclass of `SelectionModel` from package `javafx.scene.control`), which contains observable properties and various methods for manipulating the corresponding `ListView`'s items.

To respond to selection changes, you register a listener for the `MultipleSelectionModel`'s `selectedItem` property (lines 41–51). `ListView` method `getSelectionModel` returns a `MultipleSelectionModel` object. In this example, `MultipleSelectionModel`'s`selectedItemProperty` method returns a `ReadOnlyObjectProperty<Book>`, and the corresponding `ChangeListener` receives as its `oldValue` and `newValue` parameters the previously selected and newly selected `Book` objects, respectively.

Lines 47–48 use `newValue`'s large image path to initialize a new `Image` (package `javafx.scene.image`)—this loads the image from that path. We then pass the new `Image` to the

`coverImageView`'s `setImage` method to display the `Image`.