13.6 Cover Viewer App: Customizing `ListView` Cells

In the preceding example, the `ListView` displayed a `Book`’s `String` representation (i.e., its title). In this example, you’ll create a custom `ListView` cell factory to create cells that display each book as its thumbnail image and title using a `VBox`, an `ImageView` and a `Label` (Fig. 13.16).

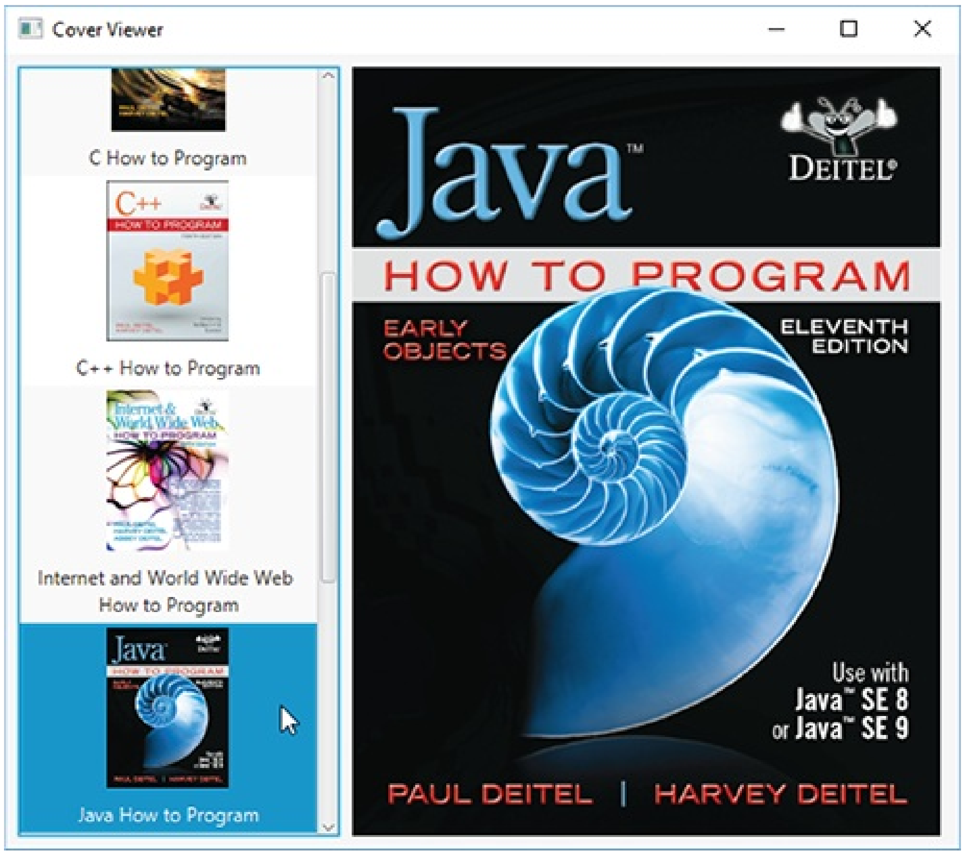

Fig. 13.16

Cover Viewer app with Java How to Program selected.

Description

13.6.1 Technologies Overview

`ListCell` Generic Class for Custom `ListView` Cell Formats

As you saw in Section 13.5, `ListView` cells display the `String` representations of a `ListView`'s items by default. To create a custom cell format, you must first define a subclass of the `ListCell` generic class (package `javafx.scene.control`) that specifies how to create a `ListView` cell. As the `ListView` displays items, it gets `ListCell`s from its cell factory. You'll use the `ListView`'s `setCellFactory` method to replace the default cell factory with one that returns objects of the `ListCell` subclass. You'll override this class's `updateItem` method to specify the cells' custom layout and contents.

Programmatically Creating Layouts and Controls

So far, you’ve created GUIs visually using JavaFX Scene Builder. In this app, you’ll also create a portion of the GUI programmatically—in fact, everything we’ve shown you in Scene Builder also can be accomplished in Java code directly. In particular, you’ll create and configure a `VBox` layout containing an `ImageView` and a `Label`. The `VBox` represents the custom `ListView` cell format.

13.6.2 Copying the `CoverViewer` App

This app’s FXML layout and classes `Book` and `CoverViewer` are identical to those in Section 13.5, and the `CoverViewerController` class has only one new statement. For this example, we’ll show a new class that implements the custom `ListView` cell factory and the one new statement in class `CoverViewerController`. Rather than creating a new app from scratch, copy the `CoverViewer` app from the previous example into a new folder named `CoverViewerCustomListView`.

13.6.3 `ImageTextCell`

Custom Cell Factory Class

Class `ImageTextCell` (Fig. 13.17) defines the custom `ListView` cell layout for this version of the **Cover Viewer** app. The class extends `ListCell<Book>` because it defines a customized presentation of a `Book` in a `ListView` cell.

```
// Fig. 13.16: ImageTextCell.java
// Custom ListView cell factory that displays an
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.scene.text.TextAlignment;

public class ImageTextCell extends ListCell<Book
   private VBox vbox = new VBox(8.0); // 8 point
   private ImageView thumbImageView = new ImageV
   private Label label = new Label();

   // constructor configures VBox, ImageView and
   public ImageTextCell() {
      vbox.setAlignment(Pos.CENTER); // center V

      thumbImageView.setPreserveRatio(true);
      thumbImageView.setFitHeight(100.0); // thu
      vbox.getChildren().add(thumbImageView); //

      label.setWrapText(true); // wrap if text t
      label.setTextAlignment(TextAlignment.CENTE
      vbox.getChildren().add(label); // attach t

      setPrefWidth(USE_PREF_SIZE); // use prefer
   }

```

```
31      // called to configure each custom ListView c
32      @Override
33      protected void updateItem(Book item, boolean
34         // required to ensure that cell displays p
35         super.updateItem(item, empty)
36
37         if (empty || item == null) {
38            setGraphic(null); // don't display anyt
39         }
40         else {
41            // set ImageView's thumbnail image
42            thumbImageView.setImage(new Image(item.
43            label.setText(item.getTitle()); // conf
44            setGraphic(vbox); // attach custom layo
45         }
46      }
47   }
```

Fig. 13.17

Custom `ListView` cell factory that displays an image and text.

Constructor

The constructor (lines 17–29) configures the instance variables we use to build the custom presentation. Line 18 indicates that the `VBox`’s children should be centered. Lines 20–22 configure the `ImageView` and attach it to the `VBox`’s collection of children. Line 20 indicates that the `ImageView` should preserve the image’s aspect ratio, and line 21 indicates that the `ImageView` should be 100 points tall. Line 22 attaches the `ImageView` to the `VBox`.

Lines 24–26 configure the `Label` and attach it to the `VBox`’s collection of children. Line 24 indicates that the `Label` should wrap its text if its too wide to fit in the `Label`’s width, and line 25 indicates that the text should be centered in the `Label`. Line 26 attaches the `Label` to the `VBox`. Finally, line 28 indicates that the cell should use its preferred width, which is determined from the width of its parent `ListView`.

Method `updateItem`

Method `updateItem` (lines 32–46) configures the `Label`’s text and the `ImageView`’s `Image` then displays the custom presentation in the `ListView`. This method is called by the `ListView`’s cell factory when a `ListView` cell is required —that is, when the `ListView` is first displayed and when `ListView` cells are about to scroll onto the screen. The method receives the `Book` to display and a `boolean` indicating whether the cell that’s about to be created is empty. You must call the superclass’s version of `updateItem` (line 35) to ensure that the custom cells display correctly.

If the cell is empty or the item parameter is `null`, then there is no `Book` to display and line 38 calls the `ImageTextCell`’s inherited `setGraphic` method with `null`. This method receives as its argument the `Node` that should be displayed in the cell. Any JavaFX `Node` can be provided, giving you tremendous flexibility for customizing a cell’s appearance.

If there is a `Book` to display, lines 40–45 configure the `ImageTextCell`’s the `Label` and `ImageView`. Line 42 configures the `Book`’s `Image` and sets it to display in the `ImageView`. Line 43 sets the `Label`’s text to the `Book`’s title. Finally, line 38 uses method `setGraphic` to set the `ImageTextCell`’s `VBox` as the custom cell’s presentation.

Performance Tip 13.1

For the best `ListView` *performance, it’s considered best practice to define the custom presentation’s controls as instance variables in the* `ListCell` *subclass and configure them in the subclass’s constructor. This minimizes the amount of work required in each call to method* `updateItem`.

13.6.4 `CoverViewerController` Class

Once you’ve defined the custom cell layout, updating the `CoverViewerController` to use it requires that you set the `ListView`’s cell factory. Insert the following code as the last statement in the `CoverViewerController`’s `initialize` method:

```
booksListView.setCellFactory(
   new Callback<ListView<Book>, ListCell<Book>>() {
      @Override
      public ListCell<Book> call(ListView<Book> listV
         return new ImageTextCell();
      }
   }
);
```

and add an import for `javafx.util.Callback`.

The argument to `ListView` method `setCellFactory` is an implementation of the functional interface `CallBack` (package `javafx.util`). This generic interface provides a `call` method that receives one argument and returns a value. In this case, we implement interface `Callback` with an object of an anonymous inner class. In `Callback`'s angle brackets the first type (`ListView<Book>`) is the parameter type for the interface's `call` method and the second (`ListCell<Book>`) is the `call` method's return type. The parameter represents the `ListView` in which the custom cells will appear. The `call` method call simply creates and returns an object of the `ImageTextCell` class.

8

Each time the `ListView` requires a new cell, the anonymous inner class's `call` method will be invoked to get a new `ImageTextCell`. Then the `ImageTextCell`'s `update` method will be called to create the custom cell presentation. Note that by using a Java SE 8 lambda (Chapter 17) rather than an anonymous inner class, you can replace the entire statement that sets the cell factory with a single line of code.