3.6 (Optional) GUI and Graphics Case Study: A Simple GUI

This case study is designed for those who want to begin learning Java's powerful capabilities for creating graphical user interfaces (GUIs) and graphics early in the book, before our deeper discussions of these topics in Chapters 12, 13 and 22. We feature **JavaFX**—Java's GUI, graphics and multimedia technology of the future. The Swing version of this case study is still available on the book's Companion Website. The GUI and Graphics Case Study sections are summarized in Fig. 3.11. Each section introduces new concepts, provides examples with screen captures that show sample interactions and is followed immediately by one or more exercises in which you'll use the techniques you learned in that section.

Section or Exercise	What you'll do
Section 3.6: A Simple GUI	Display text and an image.
Section 4.15: Event Handling; Drawing Lines	In response to a `Button` click, draw lines using JavaFX graphics capabilities.
Section 5.11: Drawing	Draw filled shapes in multiple colors.

Rectangles and Ovals	
Section 6.13: Colors and Filled Shapes	Draw filled shapes in multiple colors.
Section 7.17: Drawing Arcs	Draw a rainbow with arcs.
Section 8.16: Using Objects with Graphics	Store shapes as objects.
Section 10.14: Drawing with Polymorphism	Identify the similarities between shape classes and create a shape class hierarchy.
Exercise 13.9: Interactive Polymorphic Drawing Application	A capstone exercise in which you'll enable users to select each shape to draw, configure its properties (such as color and fill) and drag the mouse to size the shape.

Fig. 3.11

Summary of the GUI and Graphics Case Study in each chapter.

In the early GUI and Graphics Case Study sections, you'll create your first graphical apps. In subsequent sections, you'll use object-oriented programming concepts to create an app that draws a variety of shapes. The case study's capstone is Exercise 13.9 in which you'll create a GUI-based drawing application that enables the user to choose each shape to draw and its color, then specify its size and location by dragging the mouse. We hope you find this case study informative and

entertaining.

Chapter 22, JavaFX Graphics and Multimedia

If you enjoy working with GUI and graphics, be sure to read Chapter 22, which discusses JavaFX graphics and multimedia in detail, including JavaFX's powerful animation capabilities. The chapter features dozens of challenging and entertaining graphics and multimedia project exercises, including several game-programming exercises:

- **SpotOn** challenges the user to click moving spots before they disappear from the screen.
- **Horse Race** enhances the **SpotOn** game with a variation of the amusement-park horse-race game in which your horse moves in response to you rolling balls into a hole, hitting a target with a water gun, etc. In this app, the horse moves each time the user successfully clicks a spot.
- **Cannon** challenges the user to destroy a series of targets before time expires.

3.6.1 What Is a Graphical User Interface?

A **graphical user interface** (**GUI**) presents a user-friendly mechanism for interacting with an app. A GUI (pronounced "GOO-ee") gives an app a distinctive "look-and-feel." GUIs are built from **GUI components**—JavaFX refers to these as

controls. **Controls** are GUI components, such as `Label` objects that display text, `TextField` objects that enable a program to receive text typed by the user and `Button` objects that users can click to initiate actions. When the user interacts with a control, such as clicking a `Button`, the control generates an **event**. Programs can respond to these events—known as **event handling**—to specify what should happen when each user interaction occurs. This first GUI and Graphics Case Study section in which you'll build a simple non-interactive GUI is a bit longer than the others, because we walk you through every step of building your first GUI.

3.6.2 JavaFX Scene Builder and FXML

In the GUI and Graphics Case Study sections, you'll use **Scene Builder**—a tool that enables you to create GUIs simply by dragging and dropping pre-built GUI components from Scene Builder's library onto a design area, then modifying and styling the GUI *without writing any code*. JavaFX Scene Builder's live editing and preview features allow you to view your GUI as you create and modify it, without compiling and running the app. You can download Scene Builder for Windows, macOS and Linux from:

```
http://gluonhq.com/labs/scene-builder/
```

FXML (FX Markup Language)

As you create and modify a GUI, JavaFX Scene Builder generates **FXML (FX Markup Language)**—a language for defining and arranging JavaFX GUI controls without writing any Java code. JavaFX uses FXML to concisely describe GUI, graphics and multimedia elements. *You do not need to know FXML to use this case study*—Scene Builder hides the FXML details from you.

3.6.3 Welcome App—Displaying Text and an Image

In this section, *without writing any code*, you'll build a GUI that displays text in a `Label` control and an image in an `ImageView` (Fig. 3.12) control. You'll use visual-programming techniques to *drag-and-drop* JavaFX components onto Scene Builder's content panel—the design area. Next, you'll use Scene Builder's **Inspector** to configure options, such as the `Label`'s text and font size, and the `ImageView`'s image. Finally, you'll view the completed GUI using Scene Builder's **Show Preview in Window** option.

3.6.4 Opening Scene Builder and Creating the File `Welcome.fxml`

Open Scene Builder so that you can create the file into which Scene Builder will place the FXML that defines the GUI. The window initially appears as shown in Fig. 3.13. **Untitled** at the top of the window indicates that Scene Builder has created a new FXML file that you have not yet saved.2 Select **File** > **Save** to display the **Save As** dialog, then select a location in which to store the file, name the file `Welcome.fxml` and click the **Save** button.

2. We show the Scene Builder screen captures on Microsoft Windows 10, but Scene Builder is nearly identical on Windows, macOS and Linux. The key difference is that the menu bar on macOS is at the top of the screen, whereas the menu bar is part of the window on Windows and Linux.

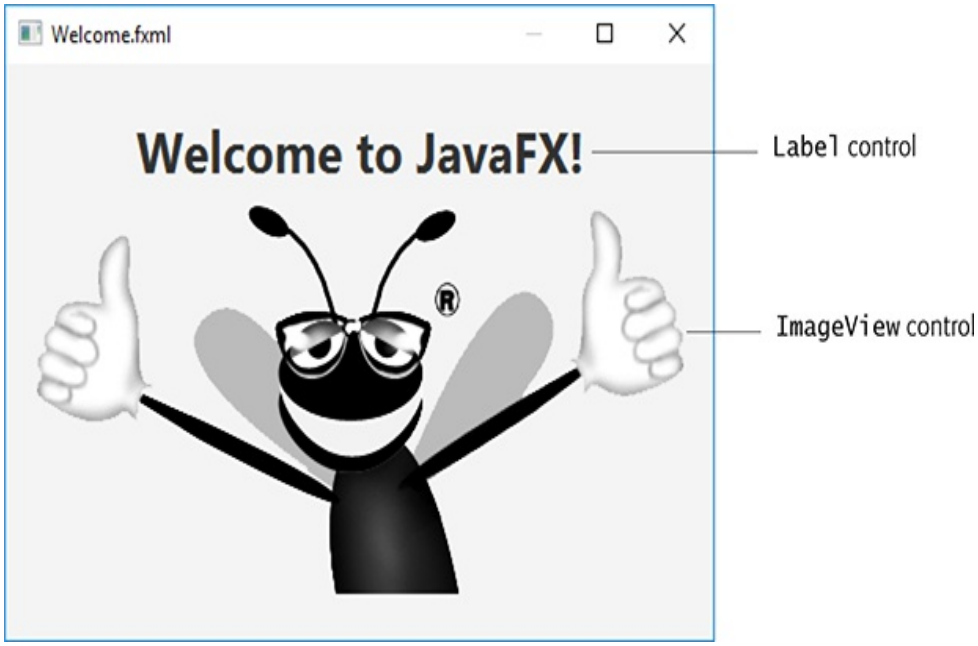

Fig. 3.12

Final **Welcome** GUI in a preview window on Microsoft Windows 10.

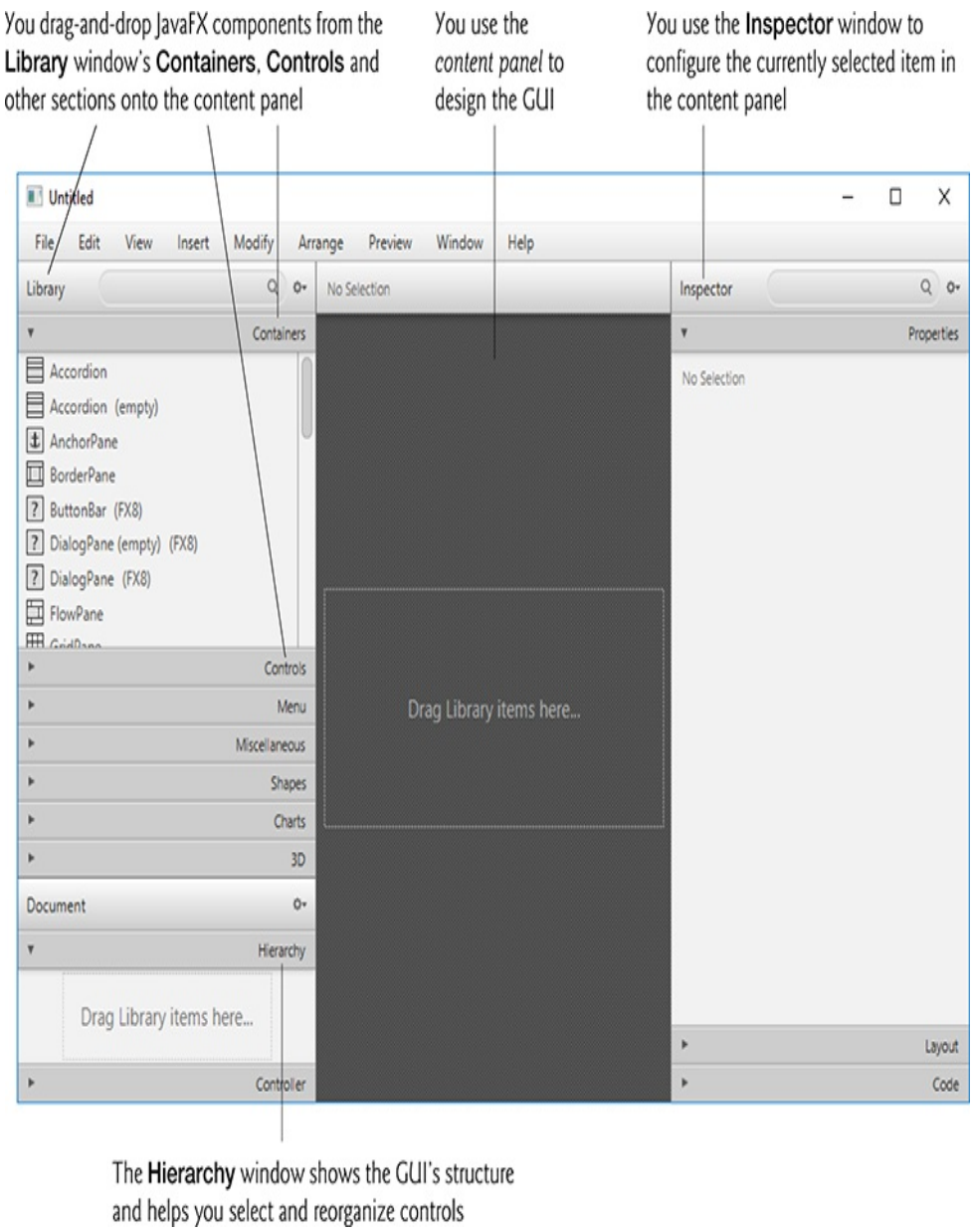

Fig. 3.13

JavaFX Scene Builder when you first open it.

Description

3.6.5 Adding an Image to the Folder Containing `Welcome.fxml`

The image you'll use for this app (`bug.png`) is located in the `images` subfolder of this chapter's `ch03` examples folder. To make it easy to find the image when you're ready to add it to the app, locate the `images` folder on your file system, then copy `bug.png` into the folder where you saved `Welcome.fxml`.

3.6.6 Creating a `VBox` Layout Container

JavaFX **Layout containers** help you arrange and size controls. For this app, you'll place a `Label` and an `ImageView` in a `VBox` **layout container**, which arranges its nodes *vertically* from top to bottom. To add a `VBox` to Scene Builder's content panel so you can begin designing the GUI, double-click **VBox** in the **Library** window's **Containers** section. (You also can

drag-and-drop a **VBox** from the **Containers** section onto the content panel.)

3.6.7 Configuring the `VBox`

In this section, you'll specify the `VBox`'s alignment, initial size and padding.

Specifying the `VBox`'s Alignment

A `VBox`'s **alignment** determines the positioning of the controls in the `VBox`. In this app, we'd like the `Label` and the `ImageView` to be centered horizontally in the `VBox`, and we'd like both to be centered vertically, so that there is an equal amount of space above the `Label` and below the `ImageView` in the final design. To accomplish this:

1. Select the `VBox` in Scene Builder's content panel by clicking it once. Scene Builder displays many `VBox` properties in the Scene Builder **Inspector**'s **Properties** section.
2. Click the down arrow for the **Alignment** property's drop-down list and notice the variety of alignment values you can use. Click `CENTER` to set the **Alignment**—this indicates that the `VBox`'s contents should be centered within the `VBox` *both horizontally and vertically*.

Each property value you specify for a JavaFX object is used to set one of that object's instance variables when JavaFX creates

the object at runtime.

Specifying the `VBox`'s Preferred Size

The **preferred size** (width and height) of a GUI's primary layout (in this case, the `VBox`) determines the default size of a JavaFX app's window. To set the preferred size:

1. Select the `VBox`.
2. Expand the **Inspector**'s **Layout** section by clicking the right arrow () next to **Layout**—this expands the section and changes the arrow to a down arrow (). Clicking the down arrow would collapse the section.
3. Click the **Pref Width** property's text field, type `450` and press *Enter* to change the preferred width.
4. Click the **Pref Height** property's text field, type `300` and press *Enter* to change the preferred height.

3.6.8 Adding and Configuring a `Label`

Next, you'll create the `Label` that displays `Welcome to JavaFX!` on the screen.

Adding a `Label` to the

VBox

Expand the Scene Builder **Library** window's **Controls** section by clicking the right arrow ( ) next to **Controls**, then drag-and-drop a **Label** from the **Controls** section onto the `VBox` in Scene Builder's content panel. (You also can double-click **Label** in the **Containers** section to add the `Label`.) Scene Builder automatically centers the `Label` object horizontally and vertically in the `VBox`, based on the `VBox`'s **Alignment** property (Fig. 3.14).

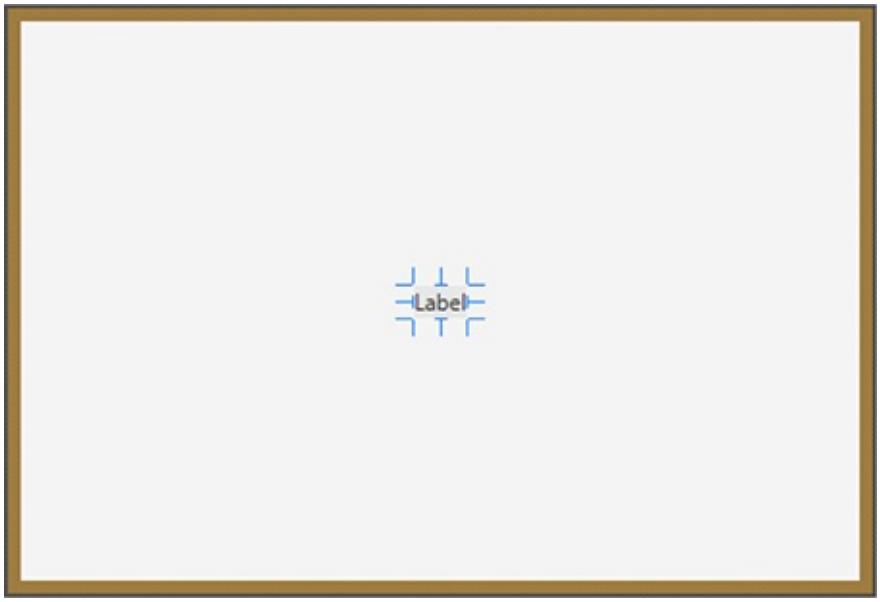

Fig. 3.14

`Label` centered in a `VBox`.

Changing the `Label`'s Text

You can set a `Label`'s text either by double clicking it and typing the new text, or by selecting the `Label` and setting its **Text** property in the **Inspector**'s **Properties** section. Set the `Label`'s text to `"Welcome to JavaFX!"`.

Changing the `Label`'s Font

For this app, we set the `Label` to display in a large bold font. To do so, select the `Label`, then in the **Inspector**'s **Properties** section, click the value to the right of the **Font** property (Fig. 3.15). In the window that appears, set the **Style** property to `Bold` and the **Size** property to `30`. The design should now appear as shown in Fig. 3.16.

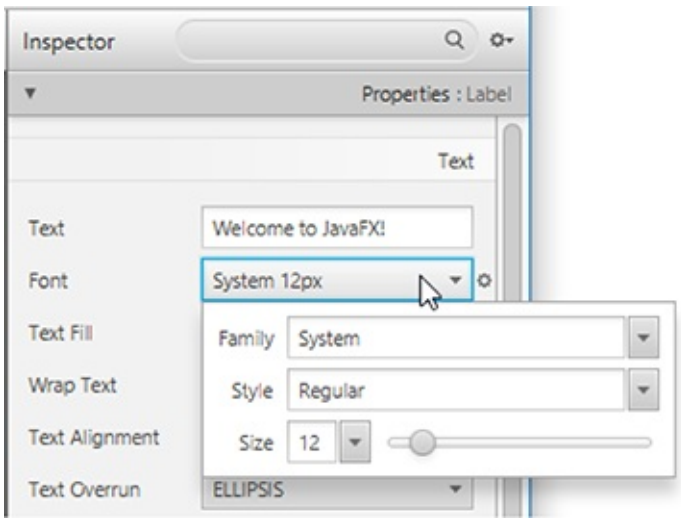

Fig. 3.15

Setting the `Label`'s **Font** property.

Description

Fig. 3.16

Welcome GUI's design after adding and configuring a `Label`.

Description

3.6.9 Adding and Configuring an `ImageView`

Finally, you'll add the `ImageView` that displays `bug.png`.

Adding an `ImageView` to the `VBox`

Drag and drop an **ImageView** from the **Library** window's **Controls** section to just below the `Label`, as shown in Fig. 3.17. You can also double-click **ImageView** in the **Library** window, in which case Scene Builder automatically places the new `ImageView` object below the `VBox`'s current contents.

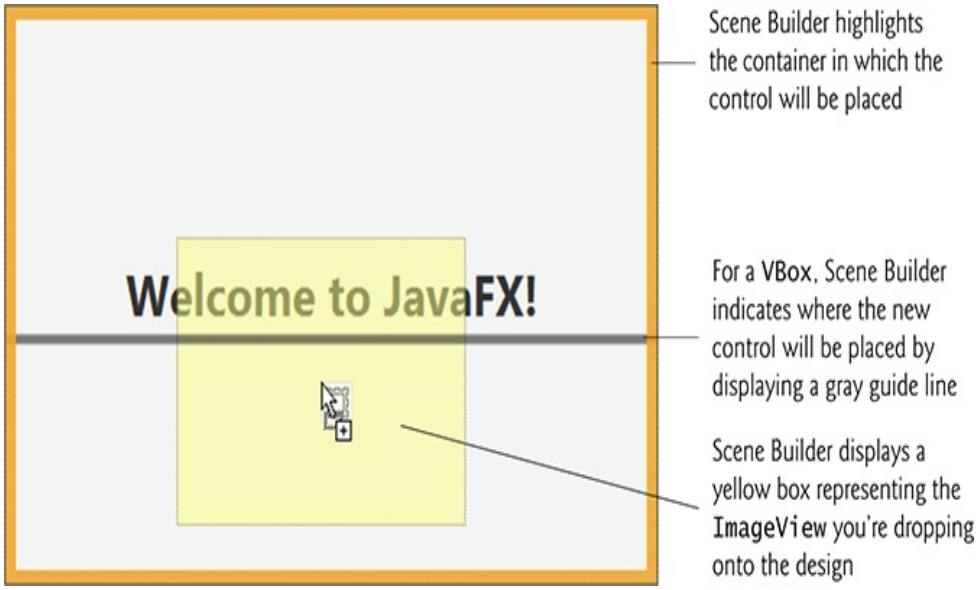

Fig. 3.17

Dragging and dropping the `ImageView` below the `Label`.

Description

Setting the `ImageView`'s Image

Next you'll set the image you'd like to display:

1. Select the `ImageView`, then in the **Inspector**'s **Properties** section click the ellipsis (**…**) button to the right of the **Image** property. By default, Scene Builder opens a dialog showing the folder in which the FXML file is saved. This is where you placed the image file `bug.png` in Section 3.6.5.
2. Select the image file, then click **Open**. Scene Builder displays the image and resizes the `ImageView` to match the image's aspect ratio—the ratio of the image's width to its height.

Changing the `ImageView`'s Size

We'd like to display the image at its original size. If you reset the `ImageView`'s default **Fit Width** and **Fit Height** property values—which Scene Builder set for you when you dragged the `ImageView` onto the design—Scene Builder will resize the `ImageView` to the image's exact dimensions. To reset these properties:

1. Expand the **Inspector**'s **Layout** section.
2. Hover the mouse over the **Fit Width** property's value. This displays the button  to the right of the property's value. Click the button and select **Reset to Default** to reset the value. This technique can be used with any property value to reset its default.

3. Repeat *Step 2* to reset the **Fit Height** property's value.

You've now completed the GUI. Scene Builder's content panel should now appear as shown in Fig. 3.18. Save the FXML file by selecting **File > Save**.

Fig. 3.18

Completed **Welcome** GUI in Scene Builder's content panel.

Description

3.6.10 Previewing the

Welcome GUI

You can preview what the design will look like in a running application's window. To do so, select **Preview > Show Preview in Window**, which displays the window in Fig. 3.19.

Fig. 3.19

Previewing the **Welcome** GUI on Microsoft Windows 10—only the window borders will differ on Linux, macOS and earlier Windows versions.

Description

GUI and Graphics Case

Study Exercise

3.1 Find four images of famous landmarks using websites such as Flickr. Create an app similar to the **Welcome** app in which you arrange the images in a collage. Add text that identifies each landmark. You can use images that are part of your project or you can specify the URL of an online image.