Chapter 12
JavaFX Graphical User Interfaces: Part 1

Java How to Program, 11/e

Questions? E-mail paul.deitel@deitel.com

OBJECTIVES

In this chapter you'll:

- Build JavaFX GUIs and handle events generated by user interactions with them.
- Understand the structure of a JavaFX app window.
- Use JavaFX Scene Builder to create FXML files that describe JavaFX scenes containing `Label`s, `ImageView`s, `TextField`s, `Slider`s and `Button`s without writing any code.

OBJECTIVES (cont.)

- Arrange GUI components using the `VBox` and `GridPane` layout containers.
- Use a controller class to define event handlers for JavaFX FXML GUI.
- Build two JavaFX apps.

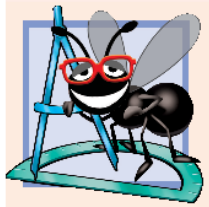
OUTLINE

OUTLINE (cont.)

OUTLINE (cont.)

Look-and-Feel Observation 12.1

Providing different apps with consistent, intuitive user-interface components gives users a sense of familiarity with a new app, so that they can learn it more quickly and use it more productively.



Software Engineering Observation 12.1

The FXML code is separate from the program logic that's defined in Java source code—this separation of the interface (the GUI) from the implementation (the Java code) makes it easier to debug, modify and maintain JavaFX GUI apps.

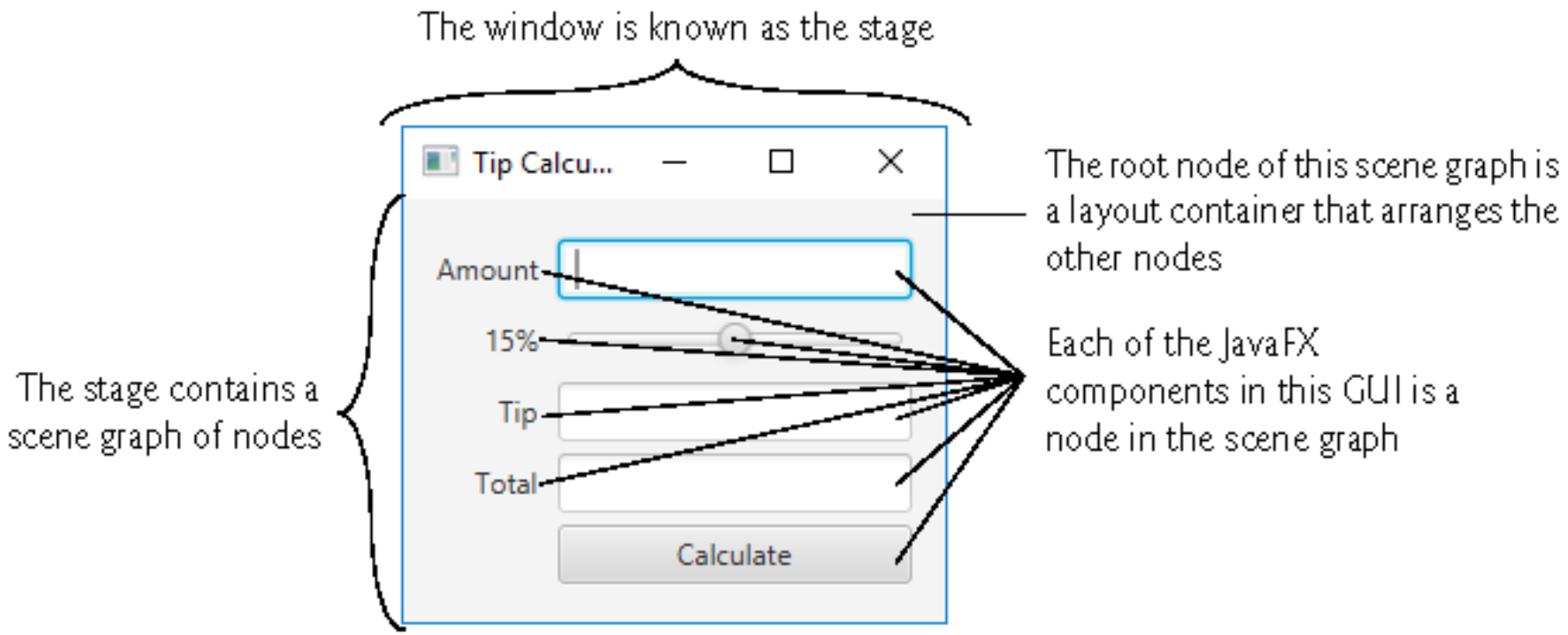


Fig. 12.1 | JavaFX app window parts.

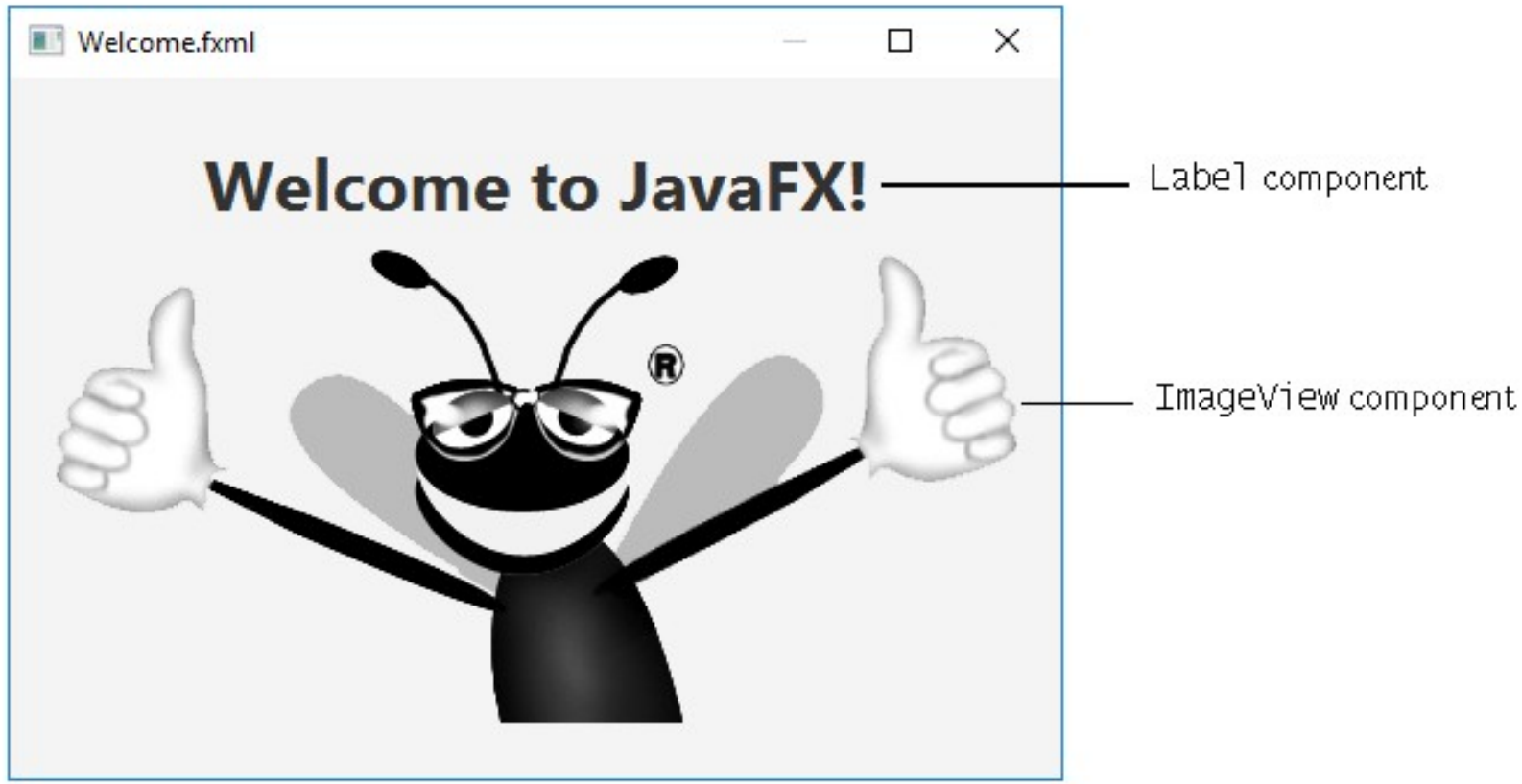


Fig. 12.2 | Final **Welcome** GUI in a preview window on Microsoft Windows 10.

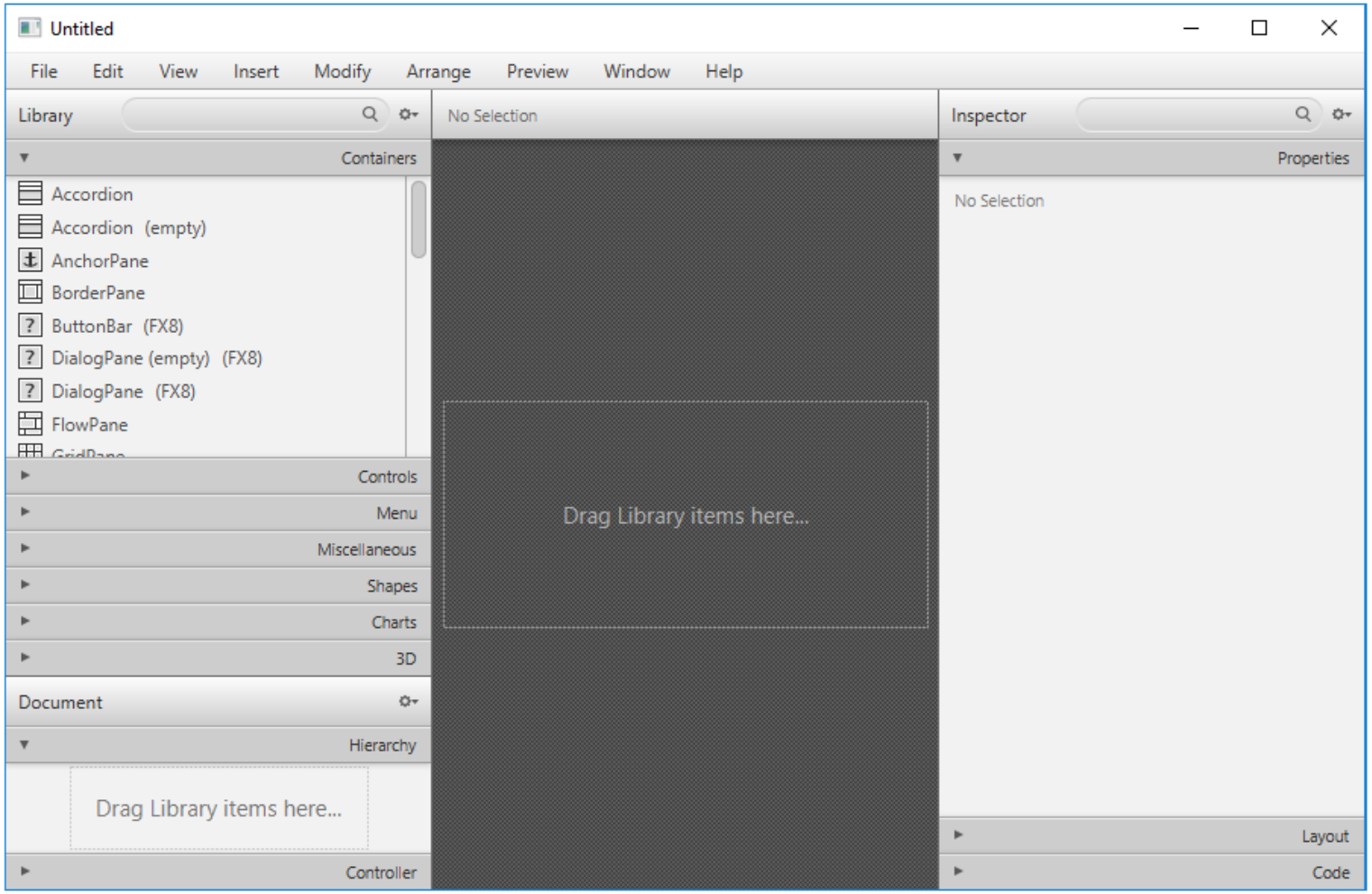


Fig. 12.3 | JavaFX Scene Builder when you first open it.

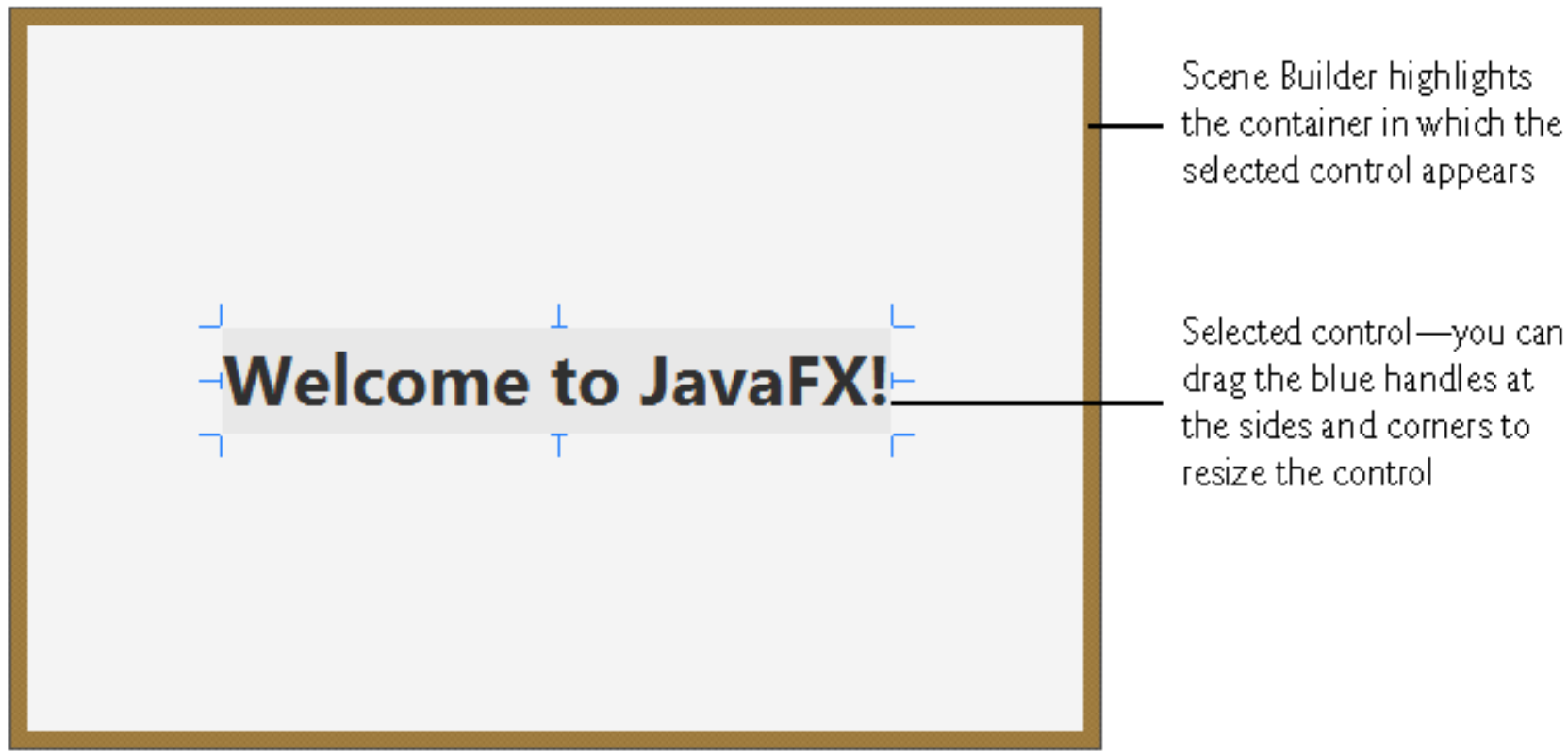


Fig. 12.4 | Welcome GUI's design after adding and configuring a **Label**.

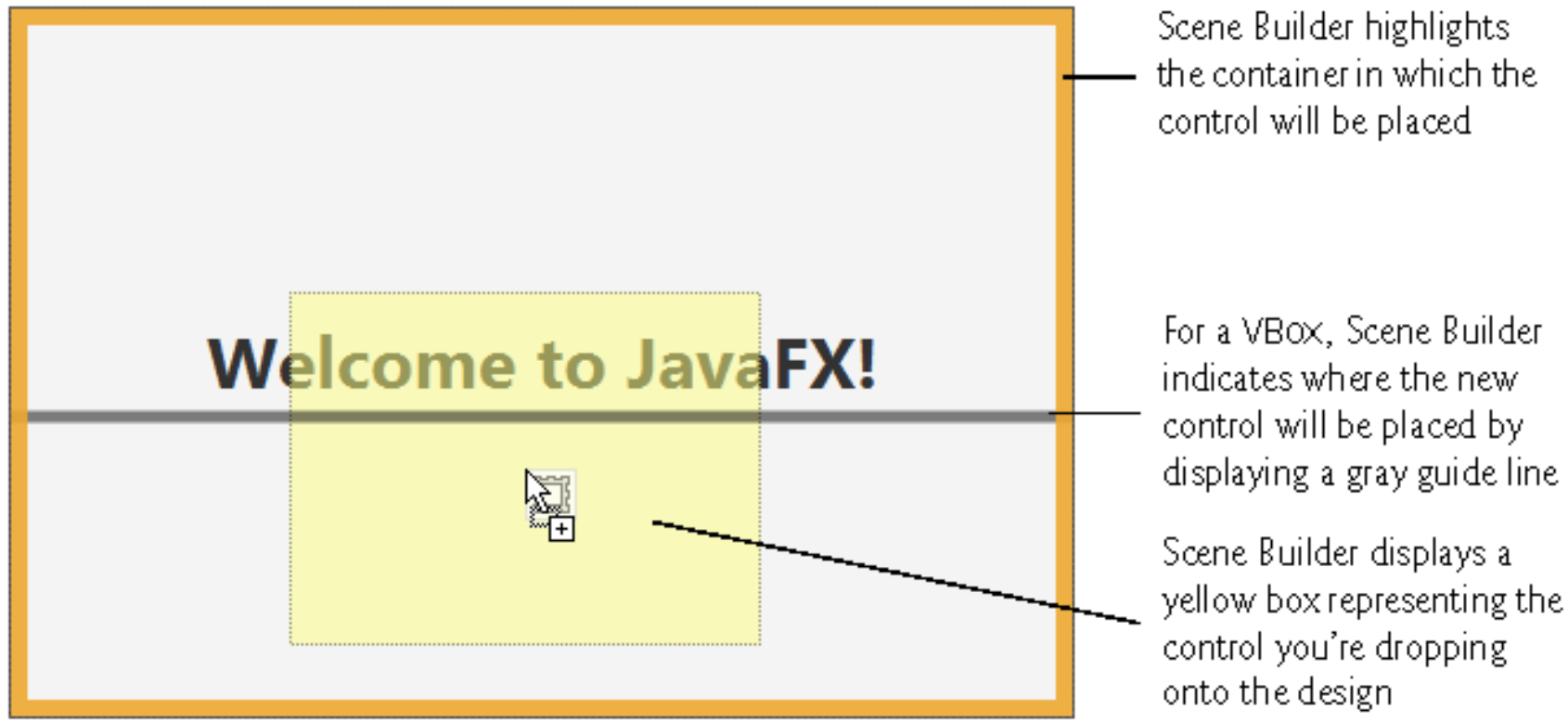


Fig. 12.5 | Dragging and dropping the `ImageView` below the `Label`.

Fig. 12.6 | Completed **Welcome** GUI in Scene Builder's content panel.

Fig. 12.7 | Previewing the **Welcome** GUI on Microsoft Windows 10—only the window borders will differ on Linux, macOS and earlier Windows versions.

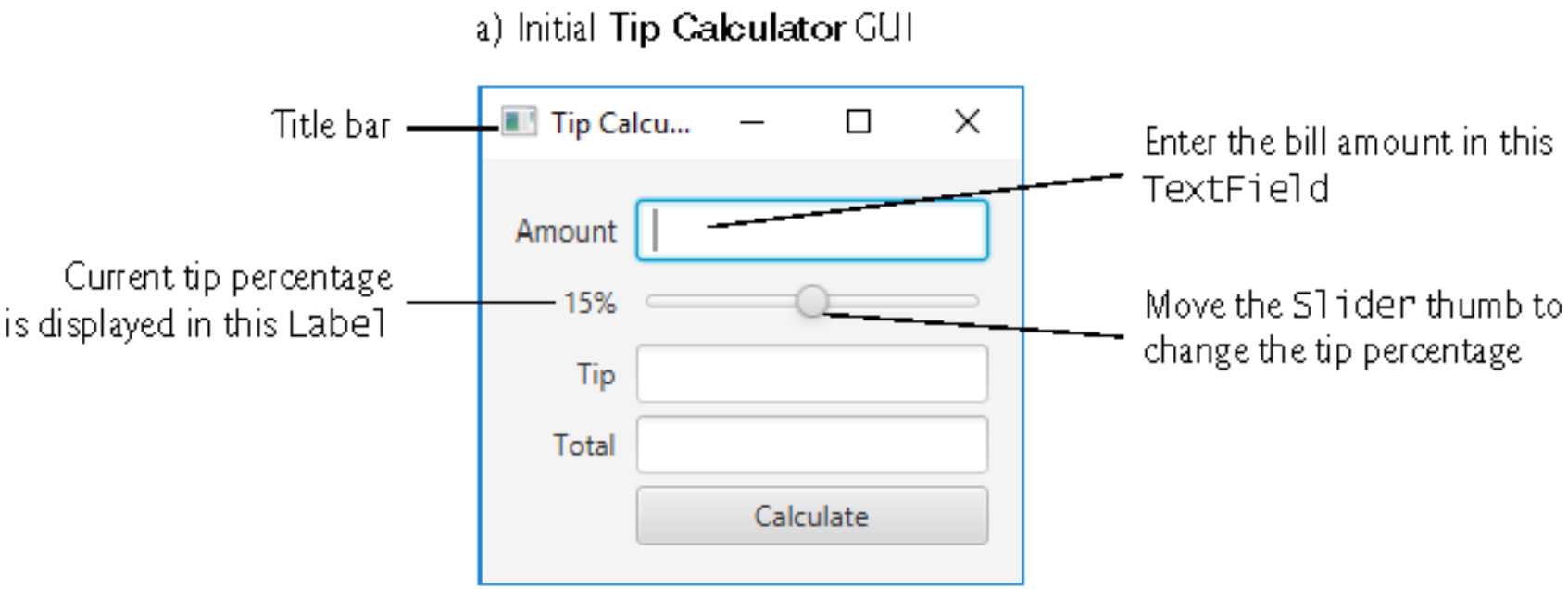


Fig. 12.8 | Entering the bill amount and calculating the tip. (Part 1 of 3.)

b) GUI after you enter the amount 34.56 and click the **Calculate** Button

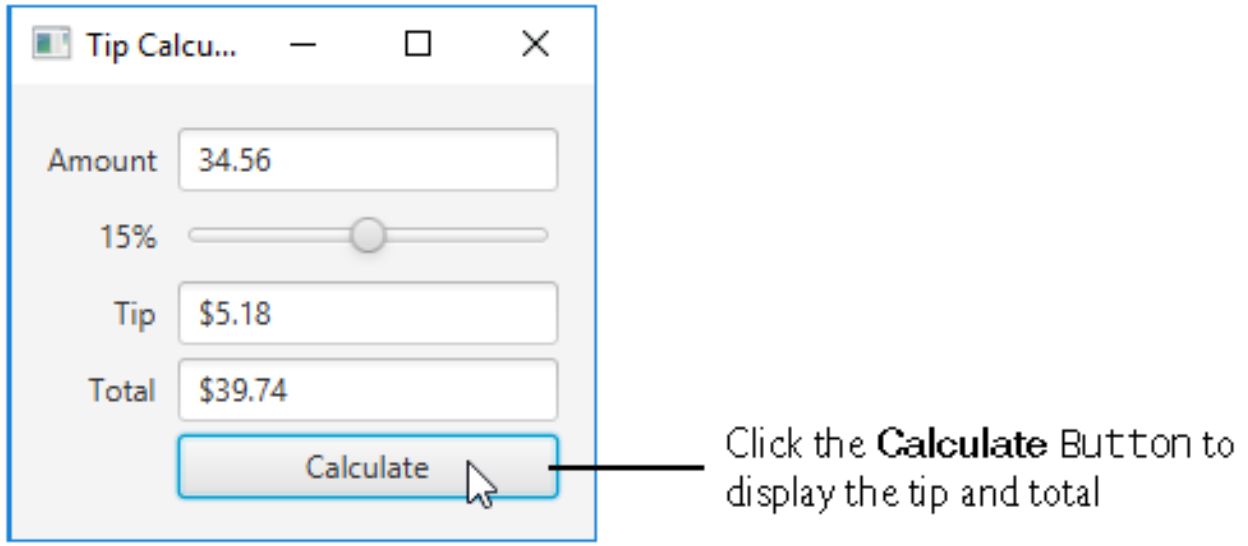


Fig. 12.8 | Entering the bill amount and calculating the tip. (Part 2 of 3.)

c) GUI after user moves the `Slider`'s thumb to change the tip percentage to 20%, then clicks the **Calculate** `Button`

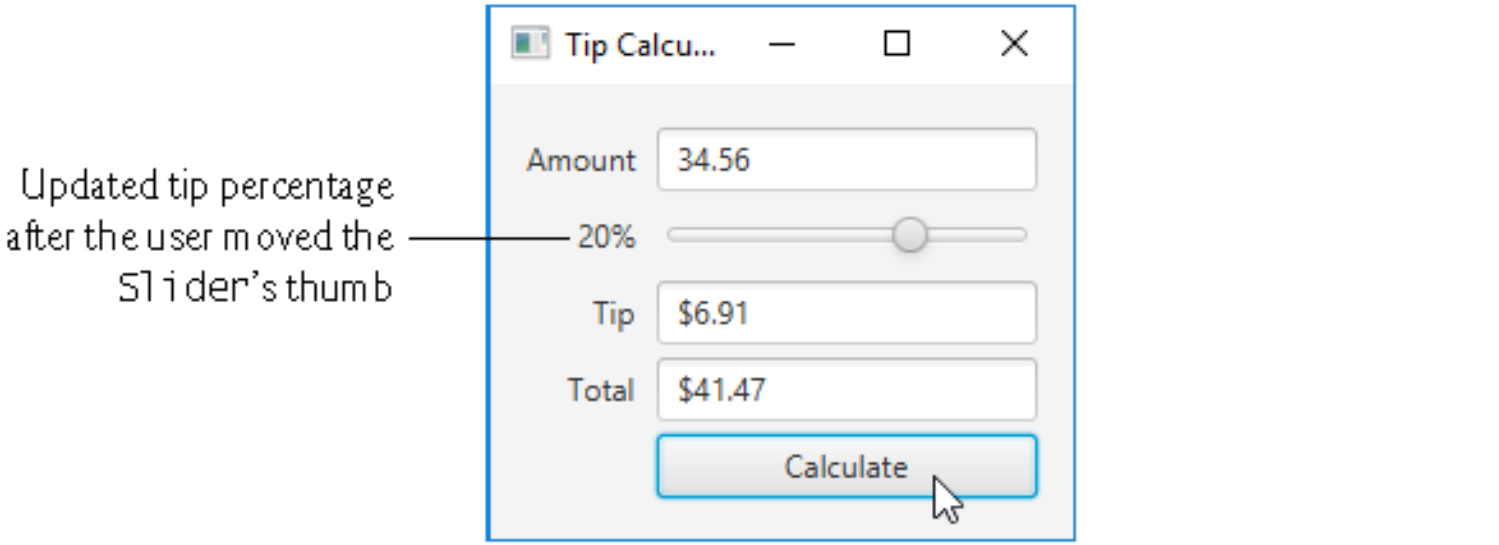


Fig. 12.8 | Entering the bill amount and calculating the tip. (Part 3 of 3.)

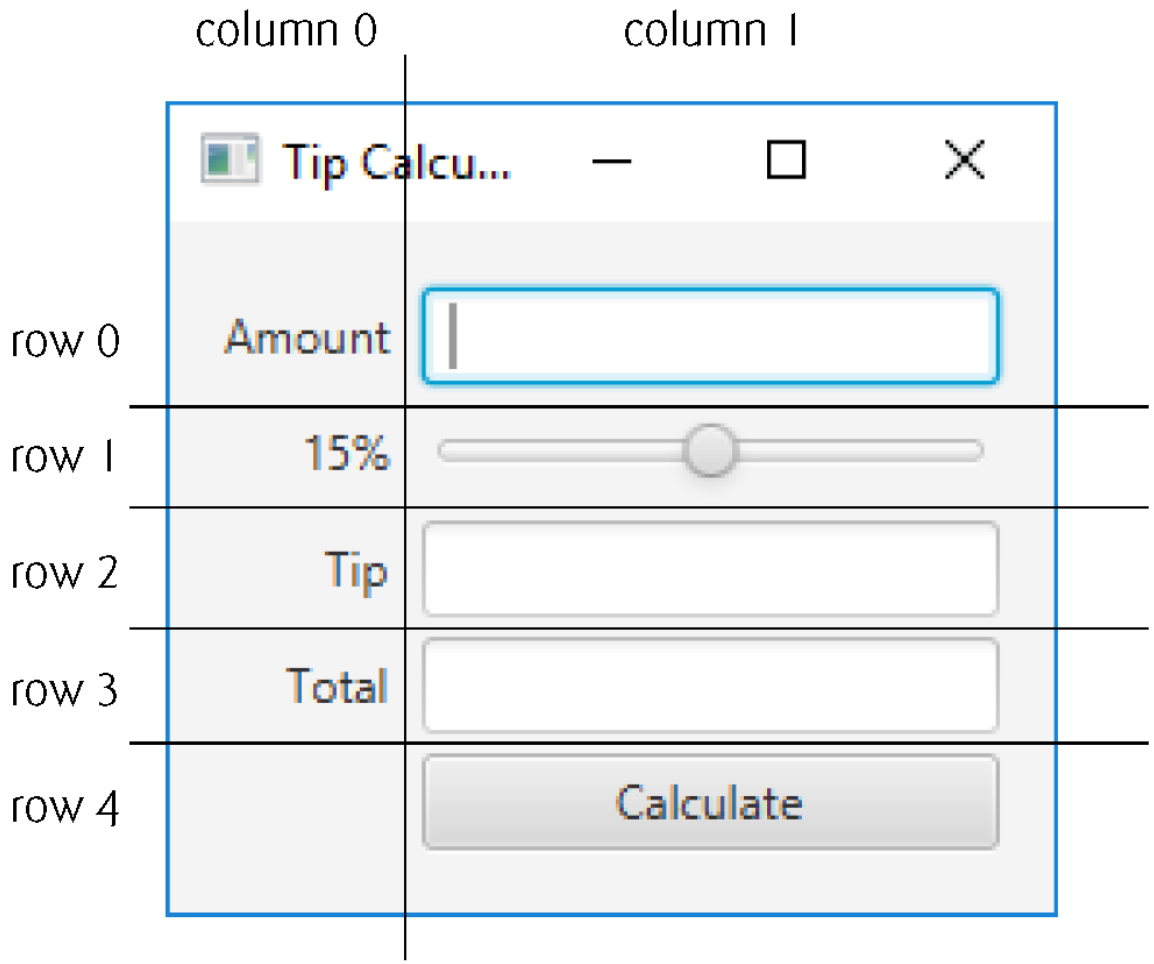


Fig. 12.9 | **Tip Calculator** GUI's `GridPane` labeled by its rows and columns.

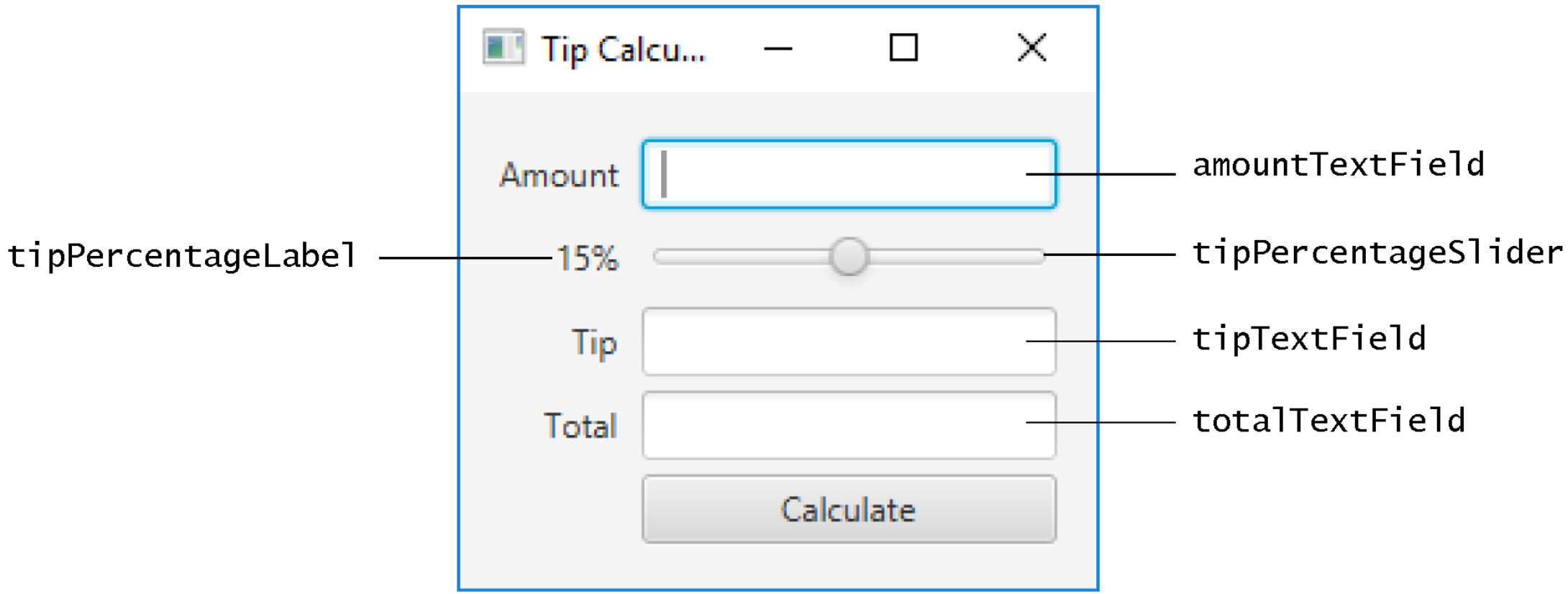


Fig. 12.10 | **Tip Calculator**'s programmatically manipulated controls labeled with their **fx:id**s.

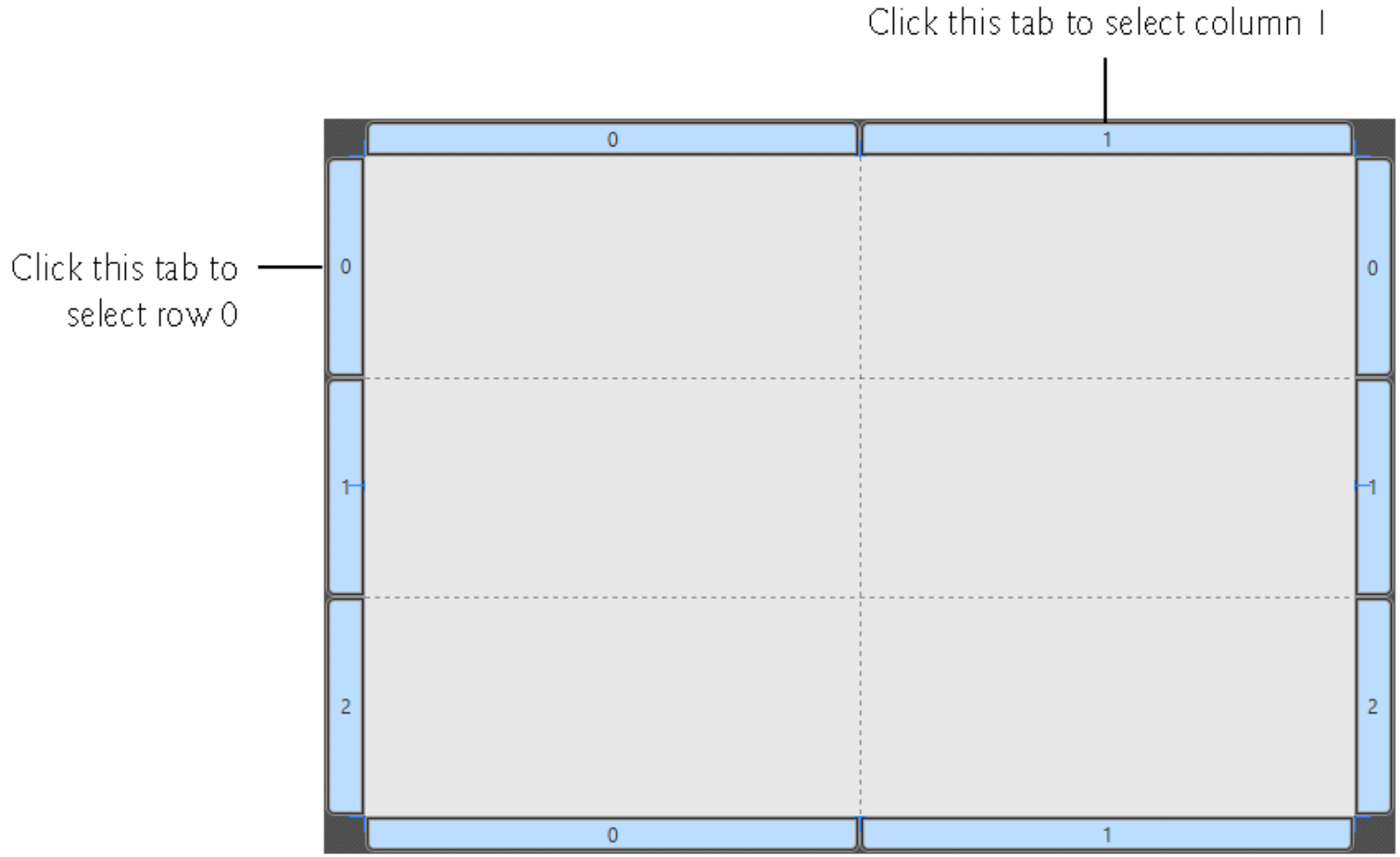


Fig. 12.11 | `GridPane` with two columns (0 and 1) and three rows (0, 1 and 2).

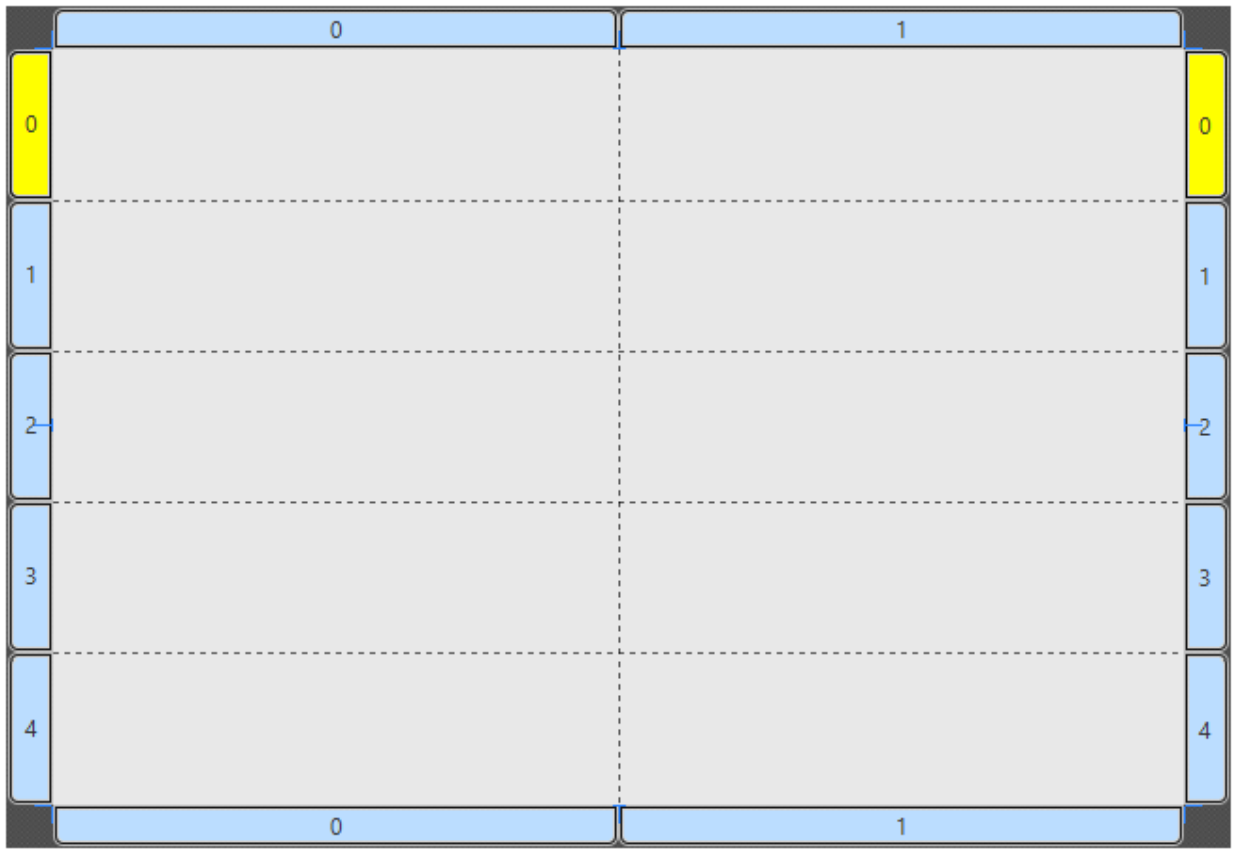


Fig. 12.12 | `GridPane` after adding two more rows.

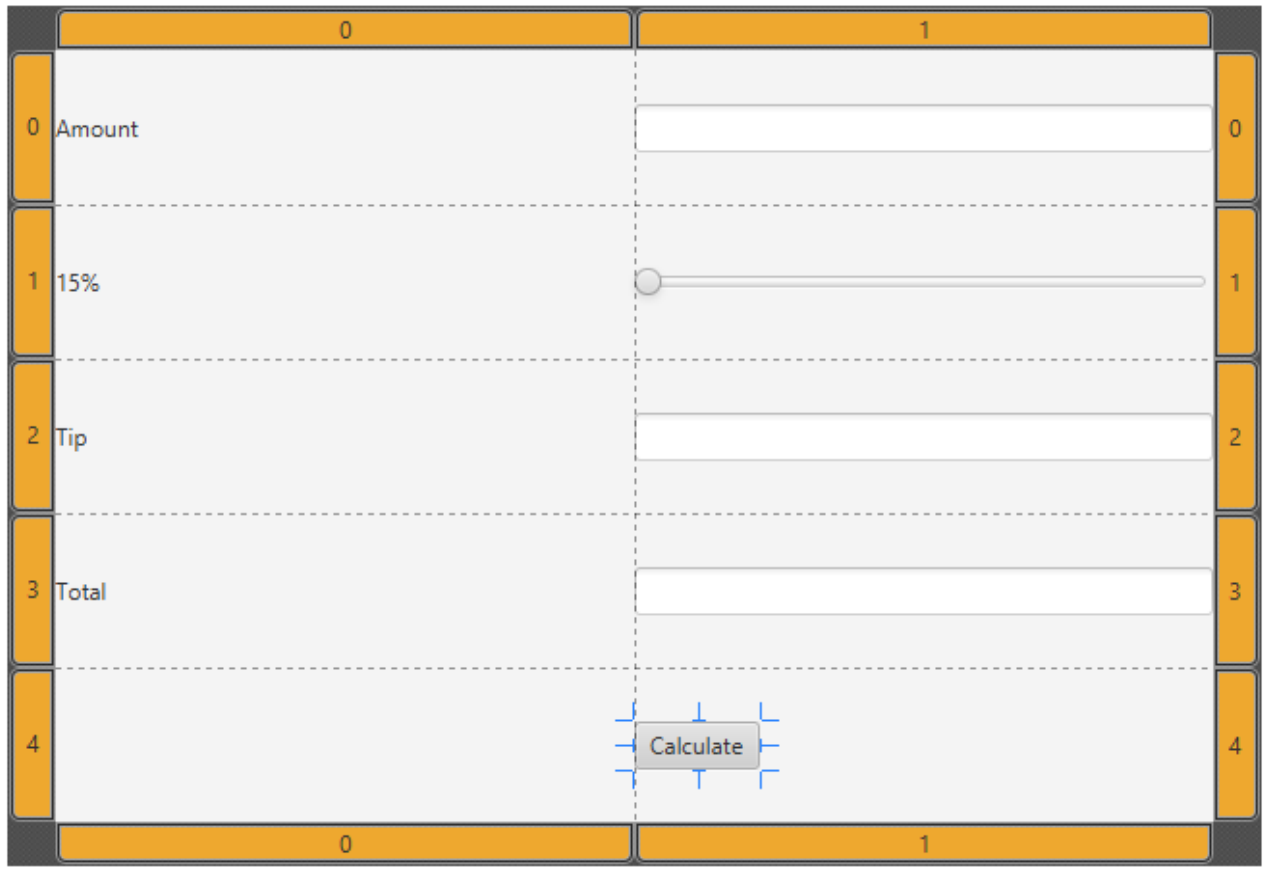


Fig. 12.13 | `GridPane` filled with the **Tip Calculator**'s controls.

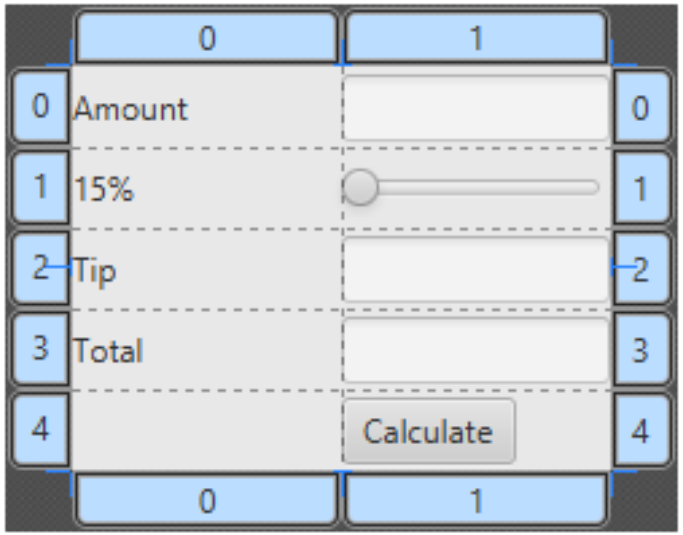


Fig. 12.14 | `GridPane` sized to fit its contents.

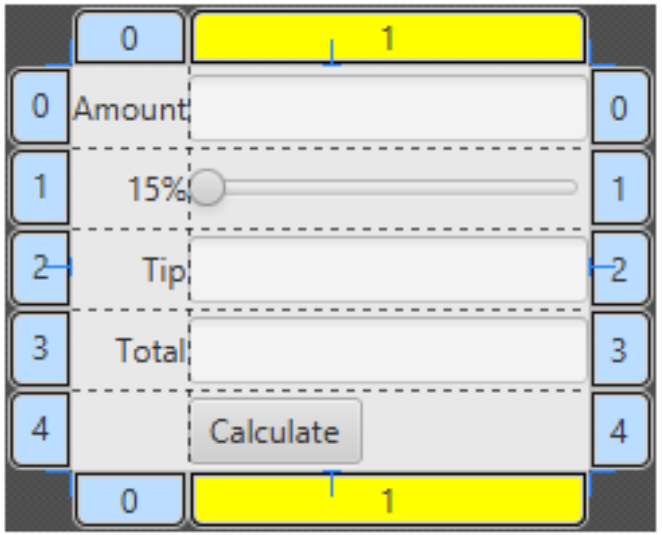


Fig. 12.15 | `GridPane` with columns sized to fit their contents.

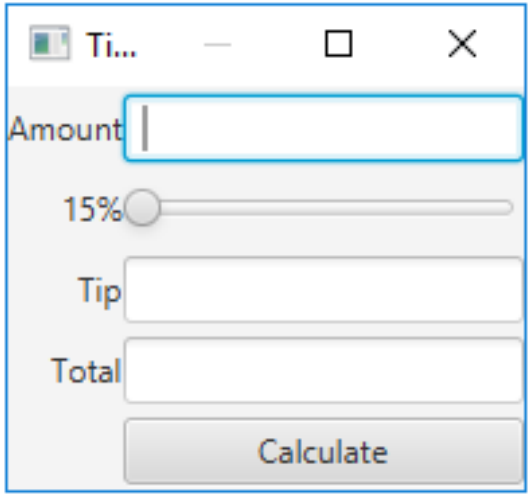


Fig. 12.16 | `GridPane` with the `TextField`s and `Button` resized.

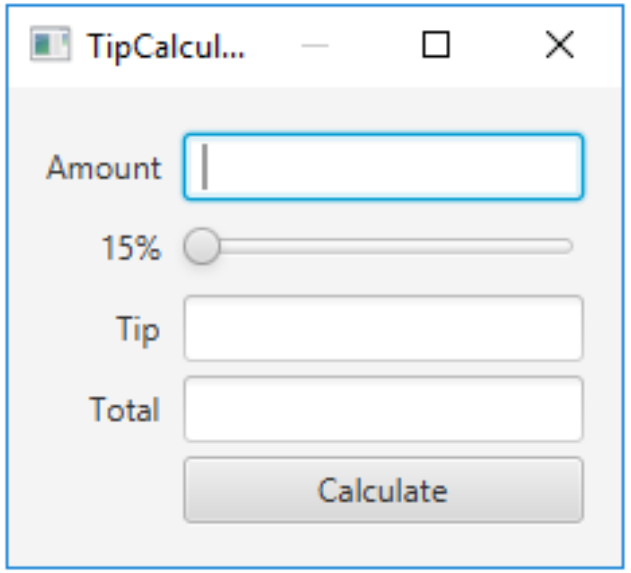


Fig. 12.17 | Final GUI design previewed in Scene Builder.

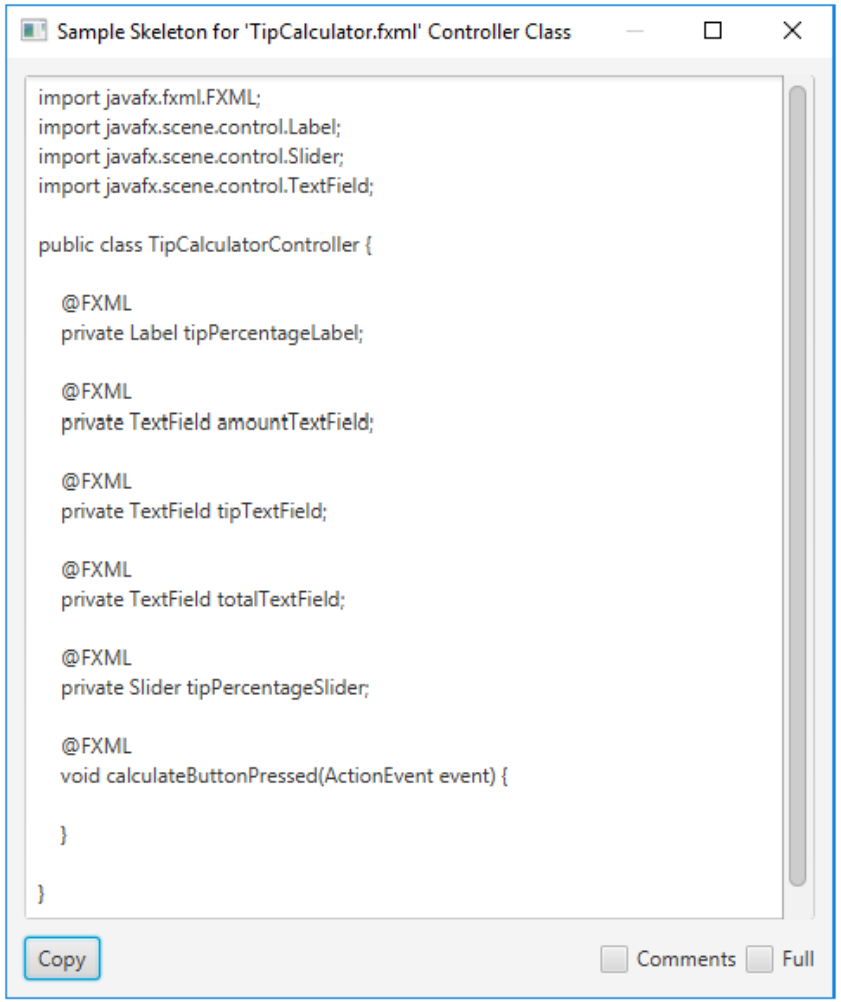


Fig. 12.18 | Skeleton code generated by Scene Builder.

```
1  // Fig. 12.19: TipCalculator.java
2  // Main app class that loads and displays the Tip Calculator's GUI
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 TipCalculator extends Application {
10    @Override
11    public void start(Stage stage) throws Exception {
12       Parent root =
13          FXMLLoader.load(getClass().getResource("TipCalculator.fxml"));
14
```

Fig. 12.19 | Main app class that loads and displays the **Tip Calculator**'s GUI. (Part 1 of 2.)

```
15          Scene scene = new Scene(root); // attach scene graph to scene
16          stage.setTitle("Tip Calculator"); // displayed in window's title bar
17          stage.setScene(scene); // attach scene to stage
18          stage.show(); // display the stage
19       }
20
21       public static void main(String[] args) {
22          // create a TipCalculator object and call its start method
23          launch(args);
24       }
25    }
```

Fig. 12.19 | Main app class that loads and displays the **Tip Calculator**'s GUI. (Part 2 of 2.)

```
1  // TipCalculatorController.java
2  // Controller that handles calculateButton and tipPercentageSlider events
3  import java.math.BigDecimal;
4  import java.math.RoundingMode;
5  import java.text.NumberFormat;
6  import javafx.beans.value.ChangeListener;
7  import javafx.beans.value.ObservableValue;
8  import javafx.event.ActionEvent;
9  import javafx.fxml.FXML;
10 import javafx.scene.control.Label;
11 import javafx.scene.control.Slider;
12 import javafx.scene.control.TextField;
13
```

Fig. 12.20 | `TipCalculatorController`'s `import` declarations.

```
14  public class TipCalculatorController {
15     // formatters for currency and percentages
16     private static final NumberFormat currency =
17        NumberFormat.getCurrencyInstance();
18     private static final NumberFormat percent =
19        NumberFormat.getPercentInstance();
20
21     private BigDecimal tipPercentage = new BigDecimal(0.15); // 15% default
22
23     // GUI controls defined in FXML and used by the controller's code
24     @FXML
25     private TextField amountTextField;
26
```

Fig. 12.21 | TipCalculatorController's `static` and instance variables. (Part 1 of 2.)

```
27      @FXML
28      private Label tipPercentageLabel;
29
30      @FXML
31      private Slider tipPercentageSlider;
32
33      @FXML
34      private TextField tipTextField;
35
36      @FXML
37      private TextField totalTextField;
38
```

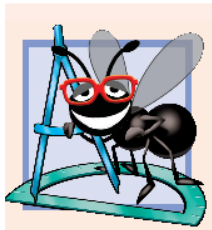
Fig. 12.21 | TipCalculatorController's `static` and instance variables. (Part 2 of 2.)

```
39      // calculates and displays the tip and total amounts
40      @FXML
41      private void calculateButtonPressed(ActionEvent event) {
42         try {
43            BigDecimal amount = new BigDecimal(amountTextField.getText());
44            BigDecimal tip = amount.multiply(tipPercentage);
45            BigDecimal total = amount.add(tip);
46
47            tipTextField.setText(currency.format(tip));
48            totalTextField.setText(currency.format(total));
49         }
50         catch (NumberFormatException ex) {
51            amountTextField.setText("Enter amount");
52            amountTextField.selectAll();
53            amountTextField.requestFocus();
54         }
55      }
56
```

Fig. 12.22 | `TipCalculatorController`'s `calculateButtonPressed` event handler.

```
57      // called by FXMLLoader to initialize the controller
58      public void initialize() {
59         // 0-4 rounds down, 5-9 rounds up
60         currency.setRoundingMode(RoundingMode.HALF_UP);
61
62         // listener for changes to tipPercentageSlider's value
63         tipPercentageSlider.valueProperty().addListener(
64            new ChangeListener<Number>() {
65               @Override
66               public void changed(ObservableValue<? extends Number> ov,
67                  Number oldValue, Number newValue) {
68                  tipPercentage =
69                     BigDecimal.valueOf(newValue.intValue() / 100.0);
70                  tipPercentageLabel.setText(percent.format(tipPercentage));
71               }
72            }
73         );
74      }
75   }
```

Fig. 12.23 | `TipCalculatorController`'s `initalize` method.



Software Engineering Observation 12.2

The event listener for an event must implement the appropriate event-listener interface.

Common Programming Error 12.1

If you forget to register an event-handler object for a particular GUI component's event type, events of that type will be ignored.