Chapter 12
JavaFX GUI : Part 1

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 `Labels`, `ImageViews`, `TextFields`, `Sliders` and `Buttons` without writing any code.
- 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.

12.1 Introduction

- History of GUI in Java
  - AWT(Abstract Window Toolkit) was Java's original GUI library
  - Swing was added to the platform in Java SE 1.2. Until recently, Swing was the primary Java GUI technology.
  - JavaFX is Java's GUI, graphics and multimedia API of the future

12.1 Introduction(Cont.)

- Some of the benefits of JavaFX over Swing include:
  - JavaFX is easier to use.
  - look-and-feel via CSS.
  - better threading support
  - uses the GPU for hardware- accelerated rendering.
  - makes apps more intuitive and easier to use.
  - Could enhancing existing GUIs.

12.2 JavaFX Scene Builder

- The Scene Builder tool is a standalone JavaFX GUI visual layout tool that can also be used with various IDEs, including the most popular ones—Eclipse, IntelliJ IDEA and NetBeans. You can download Scene Builder at:
  - http://gluonhq.com/labs/scene-builder/

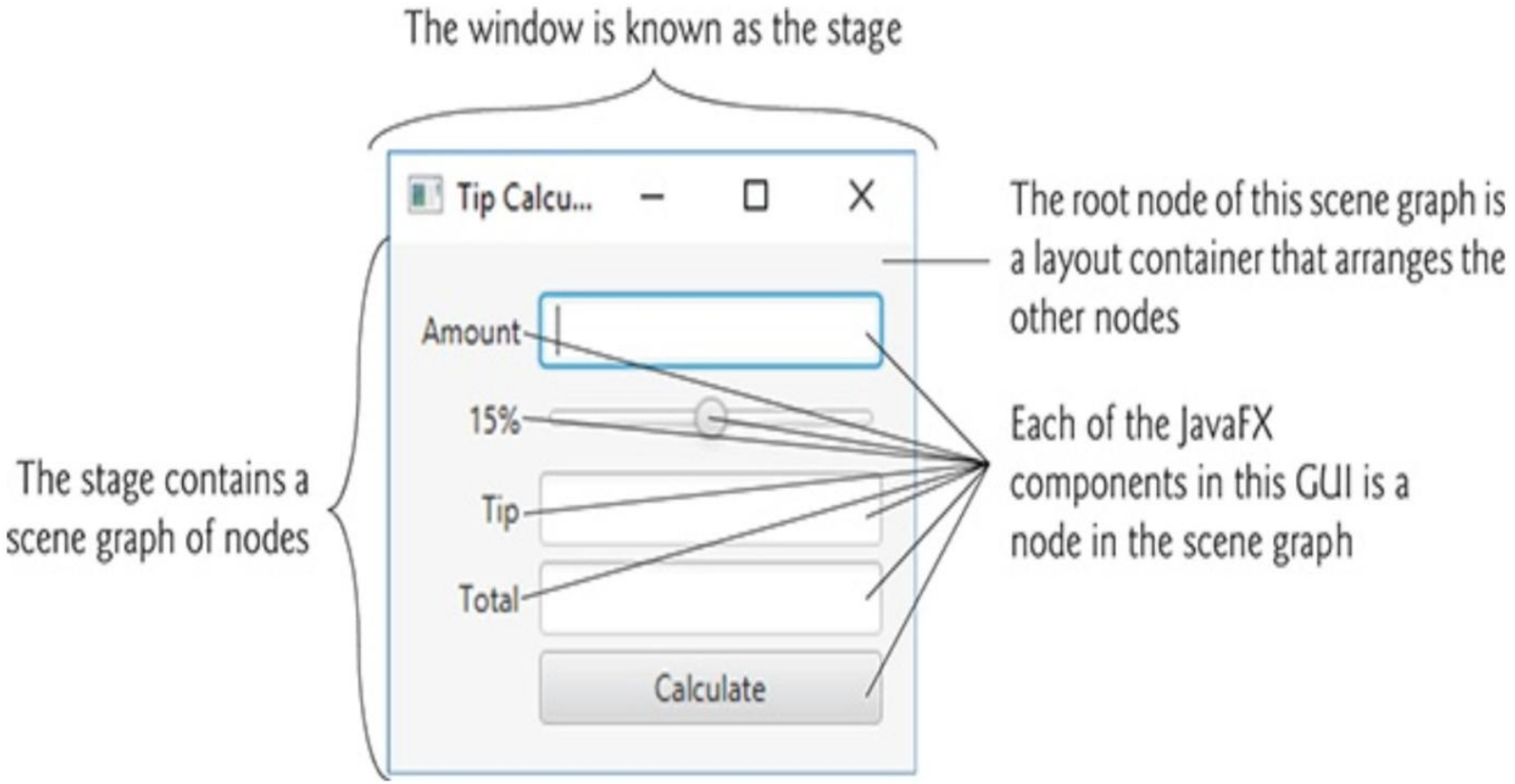
12.2 JavaFX Scene Builder(Cont.)

- You can use Cascading Style Sheets (CSS) to change the entire look-and-feel of your GUI

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.

12.3 JavaFX App Window Structure



12.3 JavaFX App Window Structure(Cont.)

- Controls
  - Controls are GUI components, such as Labels that display text, TextFields that enable a program to
  - receive user input, Buttons that users click to initiate actions, and more.
- Stage
  - The window in which a JavaFX app's GUI is displayed is known as the stage and is an instance of class Stage (package javafx.stage).
- Scene
  - The stage contains one active scene that defines the GUI as a scene graph—a tree data structure of an app's visual elements. The scene is an instance of class Scene (package javafx.scene).

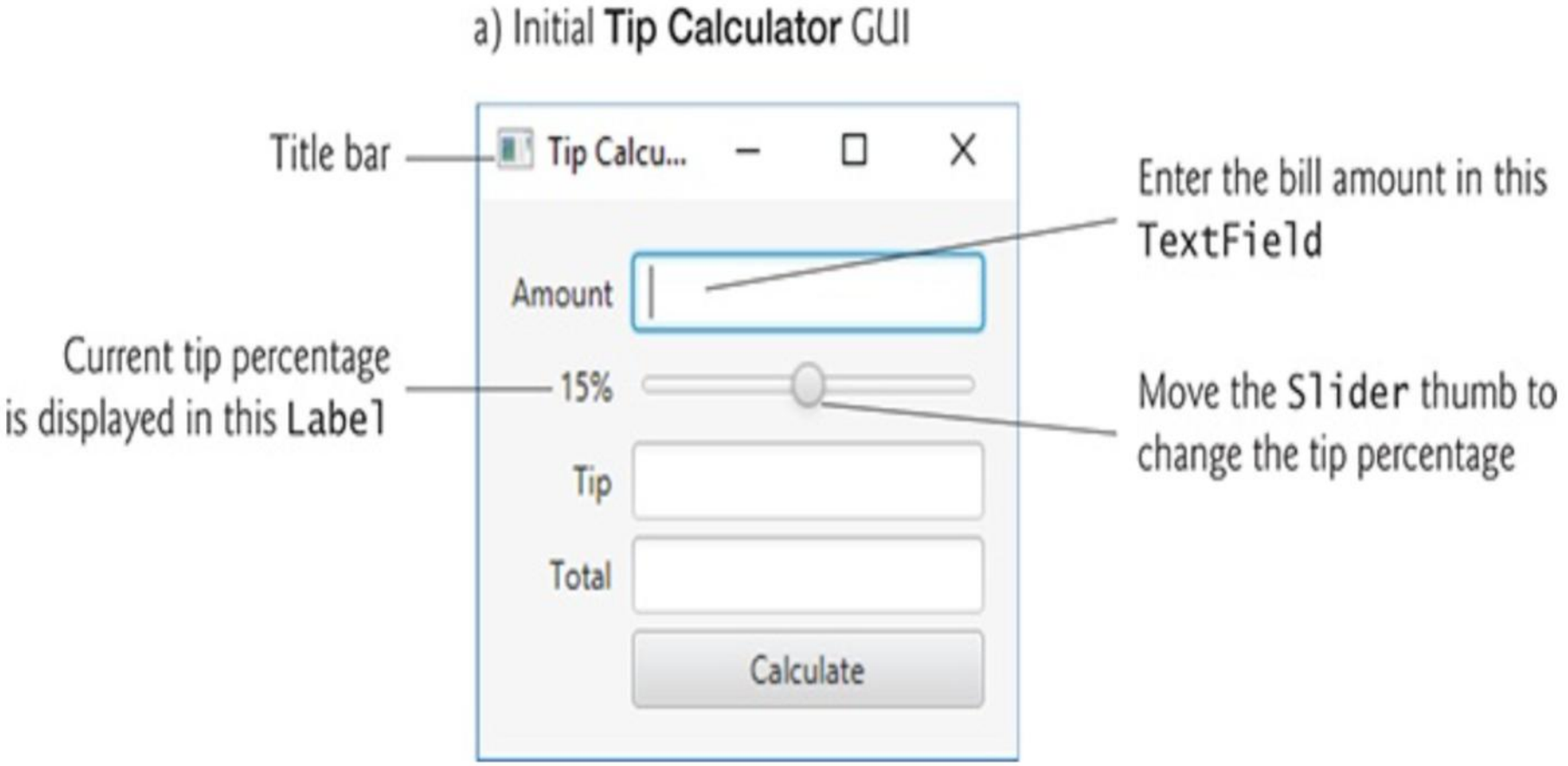
12.3 JavaFX App Window Structure(Cont.)

- Nodes
  - Each visual element in the scene graph is a node—an instance of a subclass of Node (package javafx.scene), which defines common attributes and behaviors for all nodes.
  - each node in the scene graph has one parent.
  - Nodes can have transforms (e.g., moving, rotating and scaling), opacity (whether a node is transparent, partially transparent or opaque), effects (e.g., drop shadows, blurs, reflection and lighting).

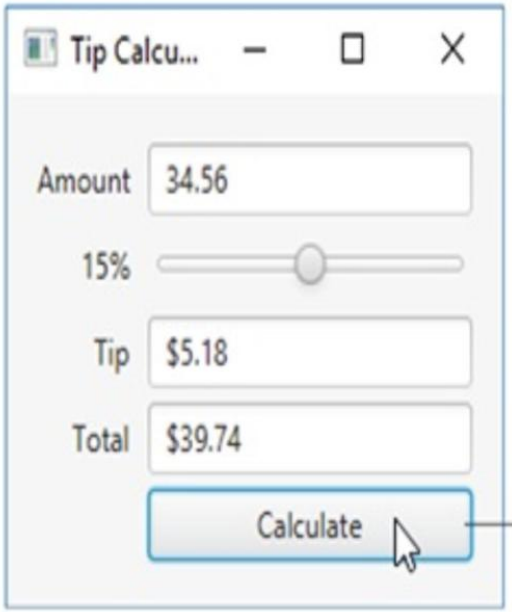
12.3 JavaFX App Window Structure(Cont.)

- Layout Containers
  - Nodes that have children are typically layout containers that arrange their child nodes in the scene
- Event Handler and Controller Class
  - An event handler is a method that responds to a user interaction.
  - An FXML GUI's event handlers are defined in a so-called controller class

12.4 Tip Calculator App— Introduction to Event Handling



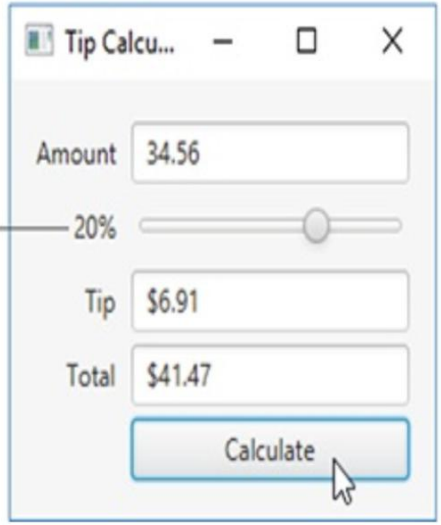
b) GUI after you enter the amount 34.56 and click the **Calculate** Button



Click the **Calculate** Button to display the tip and total

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

Updated tip percentage after the user moved the Slider's thumb



12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Class Application
  - JavaFX app is a subclass of Application (package javafx.application).
  - When the subclass's main method is called:
    - 1.Method main calls class Application's static launch method to begin executing the app.
    - 2.The launch method, in turn, causes the JavaFX runtime to call its init method, start method, and stop method.
    - 3. The Application subclass's start method creates the GUI, attaches it to a Scene and places it on the Stage that start receives as an argument.

- TestStage.java

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- GridPane
  - A GridPane (package javafx.scene.layout) arranges JavaFX components into columns and rows in a rectangular grid.
  - Each cell can be empty or can hold one or more JavaFX components
  - Each component in a GridPane can span multiple columns or rows
  - Scene Builder creates the GridPane with two columns and three rows by default.
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/GridPane.html

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- TextField
  - A TextField (package javafx.scene.control) can accept text input from the user or display text.
- Slider
  - A Slider (package javafx.scene.control) represents a value in the range 0.0–100.0 by default and allows the user to select a number in that range by moving the Slider's thumb.
- Button
  - A Button(package javafx.scene.control) allows the user to initiate an action

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Formatting Numbers as Locale-Specific Currency and Percentage Strings
  - class NumberFormat (package java.text)

```
private static final NumberFormat currency =
   NumberFormat.getCurrencyInstance();
private static final NumberFormat percent =
   NumberFormat.getPercentInstance();
```

NumberFormatDemo02.java

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Event Handling
  - When the user interacts with a GUI component, the interaction—known as an event— drives the program to perform a task.
  - The code that performs a task in response to an event is called an event handler, and the process of responding to events is known as event handling.

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Before an app can respond to an event for a particular control, you must:
  - Define an event handler that implements an appropriate interface— known as an event-listener interface.
  - Indicate that an object of that class should be notified when the event occurs—known as registering the event handler.

```
// listener for changes to tipPercentageSlider's value
tipPercentageSlider.valueProperty().addListener(
   new ChangeListener<Number>() {
      @Override
      public void changed(ObservableValue<? extends Number> ov,
         Number oldValue, Number newValue) {
         tipPercentage =
            BigDecimal.valueOf(newValue.intValue() / 100.0);
         tipPercentageLabel.setText(percent.format(tipPercentage));
      }
   }
);
```

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Using an Anonymous Inner Class for Event Handling
  - declares the event listener's class,
  - creates an object of that class and
  - registers it as the listener for changes to the tipPercentageSlider's value.
- An anonymous inner class can access its top-level class's instance variables, static variables and methods
- an anonymous inner class can access only the final or effectively final (Java SE 8,) local variables declared in the enclosing method's body.
- A variable or parameter whose value is never changed after it is initialized is effectively final.

12.4 Tip Calculator App— Introduction to Event Handling(Cont.)

- Using a Lambda to Implement the ChangeListener
  - We'll show how to implement such interfaces with lambdas in Chapter 17.

- Model-View-Controller(MVC) Architecture
  - separates an app's data (contained in the model) from the app's GUI (the view) and the app's processing logic (the controller).

```
3  // Fig. 12.19: TipCalculator.java
4  // Main application class that loads and displays the Tip Calculator's GUI.
5  import javafx.application.Application;
6  import javafx.fxml.FXMLLoader;
7  import javafx.scene.Parent;
8  import javafx.scene.Scene;
9  import javafx.stage.Stage;
10
11 public class TipCalculator extends Application {
12    @Override
13    public void start(Stage stage) throws Exception {
14       Parent root =
15          FXMLLoader.load(getClass().getResource("TipCalculator.fxml"));
16
17       Scene scene = new Scene(root); // attach scene graph to scene
18       stage.setTitle("Tip Calculator"); // displayed in window's title bar
19       stage.setScene(scene); // attach scene to stage
20       stage.show(); // display the stage
21    }
22
23    public static void main(String[] args) {
24       // create a TipCalculator object and call its start method
25       launch(args);
26    }
27 }
28
```

```
 5 import java.math.BigDecimal;
 6 import java.math.RoundingMode;
 7 import java.text.NumberFormat;
 8 import javafx.beans.value.ChangeListener;
 9 import javafx.beans.value.ObservableValue;
10 import javafx.event.ActionEvent;
11 import javafx.fxml.FXML;
12 import javafx.scene.control.Label;
13 import javafx.scene.control.Slider;
14 import javafx.scene.control.TextField;
15
16 public class TipCalculatorController {
17    // formatters for currency and percentages
18    private static final NumberFormat currency =
19       NumberFormat.getCurrencyInstance();
20    private static final NumberFormat percent =
21       NumberFormat.getPercentInstance();
22
23    private BigDecimal tipPercentage = new BigDecimal(0.15); // 15% default
24
25    // GUI controls defined in FXML and used by the controller's code
26    @FXML
27    private TextField amountTextField;
28
29    @FXML
30    private Label tipPercentageLabel;
31
32    @FXML
33    private Slider tipPercentageSlider;
34
35    @FXML
36    private TextField tipTextField;
37
38    @FXML
39    private TextField totalTextField;
40
```

```
41      // calculates and displays the tip and total amounts
42      @FXML
43      private void calculateButtonPressed(ActionEvent event) {
44         try {
45            BigDecimal amount = new BigDecimal(amountTextField.getText());
46            BigDecimal tip = amount.multiply(tipPercentage);
47            BigDecimal total = amount.add(tip);
48
49            tipTextField.setText(currency.format(tip));
50            totalTextField.setText(currency.format(total));
51         }
52         catch (NumberFormatException ex) {
53            amountTextField.setText("Enter amount");
54            amountTextField.selectAll();
55            amountTextField.requestFocus();
56         }
57      }
58
59      // called by FXMLLoader to initialize the controller
60      public void initialize() {
61         // 0-4 rounds down, 5-9 rounds up
62         currency.setRoundingMode(RoundingMode.HALF_UP);
63
64         // listener for changes to tipPercentageSlider's value
65         tipPercentageSlider.valueProperty().addListener(
66            new ChangeListener<Number>() {
67               @Override
68               public void changed(ObservableValue<? extends Number> ov,
69                  Number oldValue, Number newValue) {
70                  tipPercentage =
71                     BigDecimal.valueOf(newValue.intValue() / 100.0);
72                  tipPercentageLabel.setText(percent.format(tipPercentage));
73               }
74            }
75         );
76      }
77   }
78
```

Event Handling

- EventHandleTest.java　EventHandleMenu.java

用户行为	事件类型	类
键盘按键	KeyEvent	Node、Scene
鼠标移动或按键	MouseEvent	Node、Scene
以交替方式输入字符，或生成、改变、删除或提交	InputMethodEvent	Node、Scene
拖动对象	DragEvent	Node、Scene
滚动对象	ScrollEvent	Node、Scene
按按钮或选中菜单项	ActionEvent	ButtonBase,COntextMenu MenuItem,TextField
编辑清单、表或树的项	ListView.EditEvent; TableColumn.EditEvent TreeView.EditEvent	ListView; TableColumn TreeView
播放器出错	MediaErrorEvent	MediaView
菜单展示或隐藏	Event	Menu
弹出窗口隐藏	Event	PopupWindow
Tab 选中或关闭	Event	Tab
窗口关闭、展示或隐藏	WindowEvent	Window

Event Handling

- 1. Event Source
- Button btn = new Button();
- 2. Event Handler
  - Implement EventHandler
- 3. Register
  - setOnAction
  - addEventHandler