Chapter 13
JavaFX GUI : Part 2

13.1 Introduction

- In this chapter, you'll:
  - Use additional layouts (TitledPane, BorderPane and Pane) and controls (RadioButton and ListView).
  - Handle mouse and RadioButton events.
  - Set up event handlers that respond to property changes on controls(such as the value of a Slider).
  - Display Rectangles and Circles as nodes in the scene graph.
  - Bind a collection of objects to a ListView that displays the collection's contents.
  - [redacted] appearance of a ListView's cells.

13.2 Laying Out Nodes in a Scene Graph

- A layout determines the size and positioning of nodes in the scene graph.
- Node Size
  - In general, a node's size should not be defined explicitly.
  - Doing so often creates a design that looks pleasing when it first loads, but deteriorate(恶化） when the app is resized or the content updates.

13.2 Laying Out Nodes in a Scene Graph

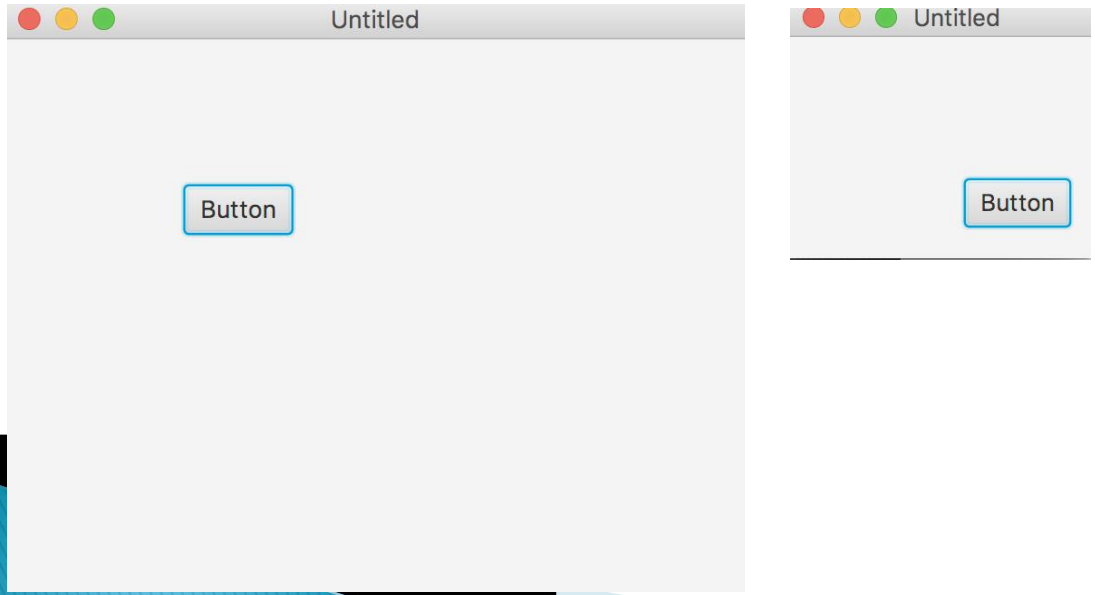
- most JavaFX nodes have the properties width, height, prefWidth, prefHeight, minWidth, minHeight, maxWidth and maxHeight
  - minimum size: a node's smallest allowed size in points.
  - maximum size:a node's largest allowed size in points.
  - preferred size :a node's preferred width and height that should be used by the layout in most cases.

13.2 Laying Out Nodes in a Scene Graph

- Pane
  - JavaFX layout panes are container nodes that arrange their child nodes in a scene graph relative to one another, based on their sizes and positions.
  - Most JavaFX layout panes use relative positioning—if a layout-pane node is resized, it adjusts its children's sizes and positions accordingly, based on their preferred, minimum and maximum sizes.

13.2 Laying Out Nodes in a Scene Graph

- AnchorPane
  - Enables you to set the position of child nodes relative to the pane's edges. Resizing the pane does not alter the layout of the nodes.



13.2 Laying Out Nodes in a Scene Graph

- Example:

```
AnchorPane anchorPane = new AnchorPane();
// List should stretch as anchorPane is resized
ListView list = new ListView();
AnchorPane.setTopAnchor(list, 10.0);
AnchorPane.setLeftAnchor(list, 10.0);
AnchorPane.setRightAnchor(list, 65.0);
// Button will float on right edge
Button button = new Button("Add");
AnchorPane.setTopAnchor(button, 10.0);
AnchorPane.setRightAnchor(button, 10.0);
anchorPane.getChildren().addAll(list, button);
```

13.2 Laying Out Nodes in a Scene Graph

- BorderPane
  - Includes five areas—top, bottom, left, center and right— where you can place nodes. The top and bottom regions fill the BorderPane's width and are vertically sized to their children's preferred heights. The left and right regions fill the BorderPane's height and are horizontally sized to their children's preferred widths. The center area occupies all of the BorderPane's remaining space. You might use the different areas for tool bars, navigation, a main content area, etc.

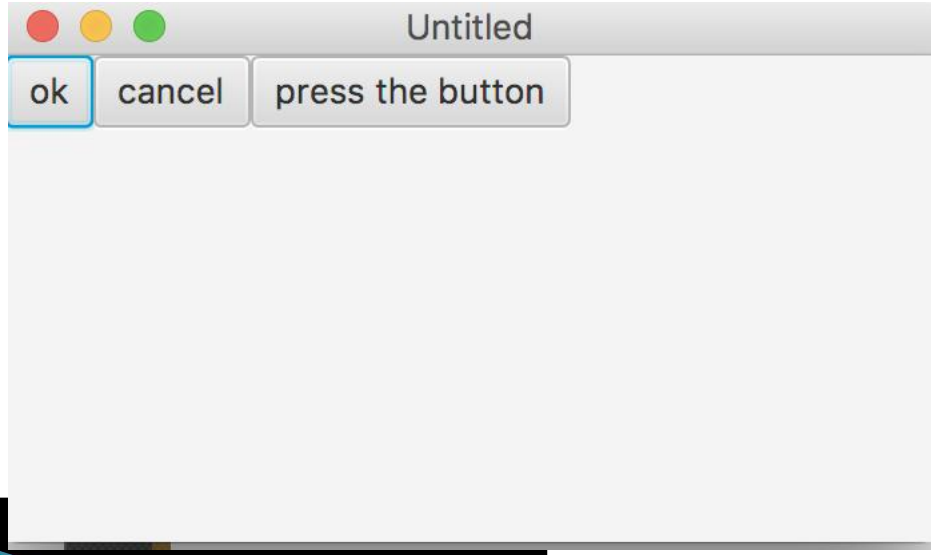
13.2 Laying Out Nodes in a Scene Graph

- Example:

```
BorderPane borderPane = new BorderPane();
ToolBar toolbar = new ToolBar();
HBox statusbar = new HBox();
Button button = new Button();
borderPane.setTop(toolbar);
borderPane.setCenter(button);
borderPane.setBottom(statusbar);
```

13.2 Laying Out Nodes in a Scene Graph

- FlowPane:
  - Lays out nodes consecutively—either horizontally or vertically. When the boundary for the pane is reached, the nodes wrap to a new line in a horizontal FlowPane or a new column in a vertical FlowPane.



13.2 Laying Out Nodes in a Scene Graph

- Example:

```
Image images[] = { ... };
FlowPane flow = new FlowPane();
flow.setVgap(8);
flow.setHgap(4);
flow.setPrefWrapLength(300); // preferred width = 300
for (int i = 0; i < images.length; i++)
{ flow.getChildren().add(new ImageView(image[i]); }
```

13.2 Laying Out Nodes in a Scene Graph

- GridPane:
  - Creates a flexible grid for laying out nodes in rows and columns.

13.2 Laying Out Nodes in a Scene Graph

- Example:

```
GridPane gridpane = new GridPane();
gridpane.add(new Button(), 1, 0); // column=1 row=0
gridpane.add(new Label(), 2, 0); // column=2 row=0
```

```
GridPane gridpane = new GridPane();
 // Set one constraint at a time...
// Places the button at the first row and second
column Button button = new Button();
GridPane.setRowIndex(button, 0);
GridPane.setColumnIndex(button, 1);
// or convenience methods set more than one
constraint at once...
Label label = new Label();
GridPane.setConstraints(label, 2, 0);
// column=2 row=0 // don't forget to add children to
gridpane
gridpane.getChildren().addAll(button, label);
```

13.2 Laying Out Nodes in a Scene Graph

- Pane:
  - The base class for layout panes. This can be used to position nodes at fixed locations—known as absolute positioning.

13.2 Laying Out Nodes in a Scene Graph

- Example:

```
Pane canvas = new Pane();
canvas.setPrefSize(200,200);
Circle circle = new Circle(50,Color.BLUE);
circle.relocate(20, 20);
Rectangle rectangle = new
Rectangle(100,100,Color.RED);
rectangle.relocate(70,70);
canvas.getChildren().addAll(circle,rectangle);
```

13.2 Laying Out Nodes in a Scene Graph

- StackPane:
  - Places nodes in a stack. Each new node is stacked atop the previous node. You might use this to place text on top of images, for example.

13.2 Laying Out Nodes in a Scene Graph

- Example:

```
// Align the title Label at the bottom-center of the stackpane
 Label title = new Label();
StackPane.setAlignment(title, Pos.BOTTOM_CENTER);
stackpane.getChildren.addAll(new ImageView(...), title);
// Create an 8 pixel margin around a listview in the
stackpane
ListView list = new ListView();
StackPane.setMargin(list, new Insets(8,8,8,8);
stackpane.getChildren().add(list);
```

13.2 Laying Out Nodes in a Scene Graph

- TilePane:
  - A horizontal or vertical grid of equally sized tiles. Nodes that are tiled horizontally wrap at the TilePane's width. Nodes that are tiled vertically wrap at the TilePane's height.

13.2 Laying Out Nodes in a Scene Graph

- Example:

```
TilePane tile = new TilePane();
tile.setHgap(8);
tile.setPrefColumns(4);
for (int i = 0; i < 20; i++)
{ tile.getChildren().add(new ImageView(...)); }
```

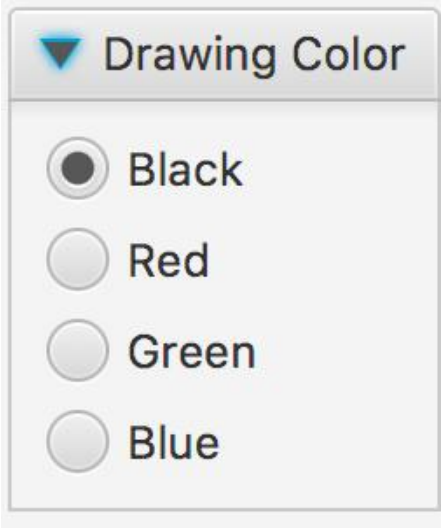
13.2 Laying Out Nodes in a Scene Graph

- Example:

```
TilePane tile = new TilePane(Orientation.VERTICAL);
tile.setTileAlignment(Pos.CENTER_LEFT);
tile.setPrefRows(10);
for (int i = 0; i < 50; i++) { tile.getChildren().add(new
Button(...)); }
```

13.2 Laying Out Nodes in a Scene Graph

- TitlePane:
  - A TitledPane is a panel with a title that can be opened and closed.



13.2 Laying Out Nodes in a Scene Graph

- Example：

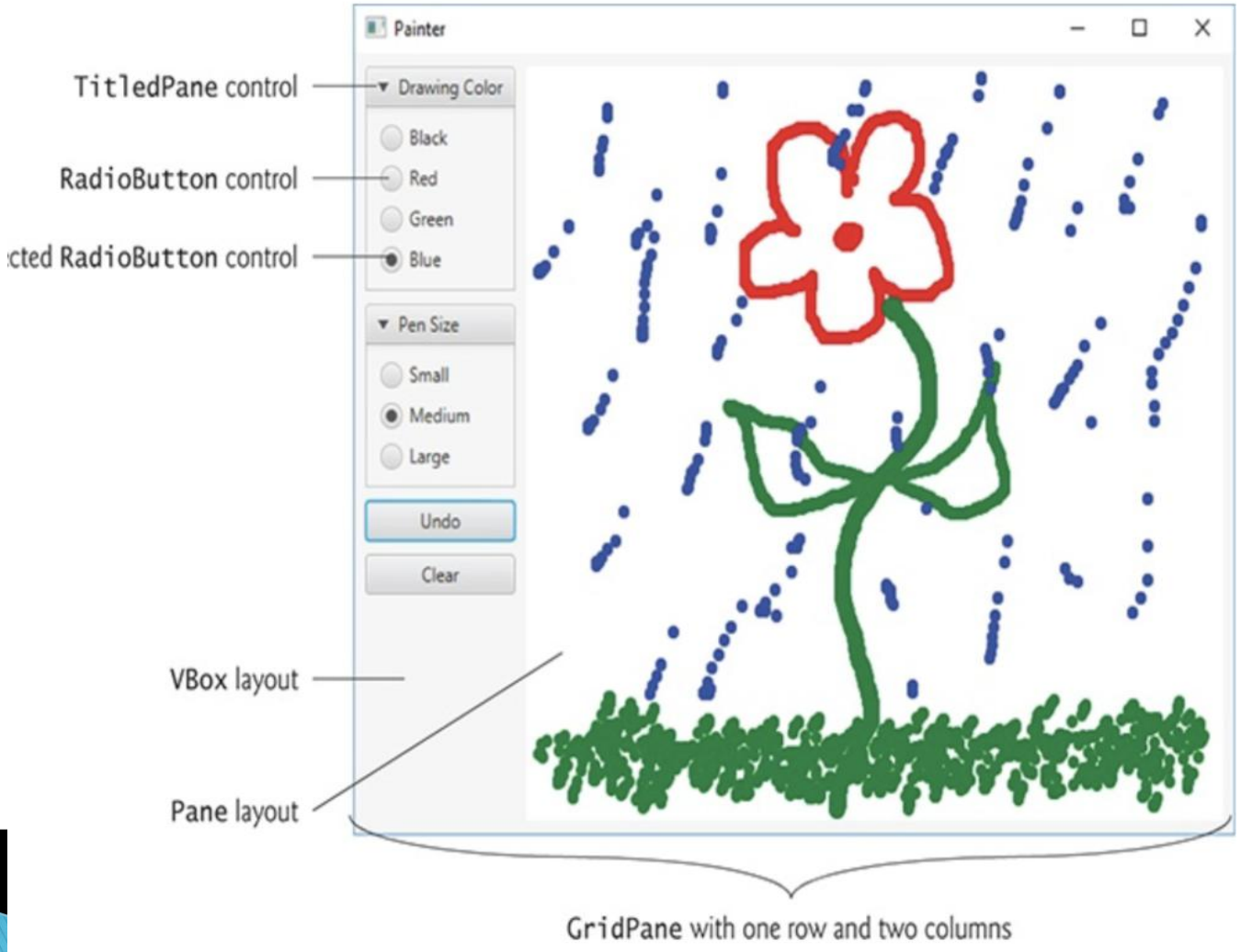
```
TitledPane titledPane = new TitledPane("My Title", new CheckBox("OK"));
```

```
final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
    final Image[] images = new Image[imageNames.length];
    final ImageView[] pics = new ImageView[imageNames.length];
    final TitledPane[] tps = new TitledPane[imageNames.length];

    @Override public void start(Stage stage) {
      ……
       final Accordion accordion = new Accordion ();
       for (int i = 0; i < imageNames.length; i++) {
           images[i] = new
               Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
           pics[i] = new ImageView(images[i]);
           tps[i] = new TitledPane(imageNames[i],pics[i]);
       }

       accordion.getPanes().addAll(tps);          在一个Accordion中同一时间只能打开一个
       accordion.setExpandedPane(tps[0]);         Titled Pane
```

13.2 Laying Out Nodes in a Scene Graph

- HBox:
  - Arranges nodes horizontally in one row.
- VBox
  - Arranges nodes vertically in one column.

13.3 Painter App: RadioButtons, Mouse Events and Shapes



13.3 Painter App: RadioButtons, Mouse Events and Shapes(Cont.)

- RadioButtons and ToggleGroups
  - RadioButtons function as mutually exclusive options.
  - You add multiple RadioButtons to a ToggleGroup to ensure that only one RadioButton in a given group is selected at a time.
- TitledPane Layout Container
  - A TitledPane layout container displays a title at its top and is a collapsible panel containing a layout node, which in turn contains other nodes.

13.3 Painter App: RadioButtons, Mouse Events and Shapes(Cont.)

- JavaFX Shapes
  - The javafx.scene.shape package contains various classes for creating 2D and 3D shape nodes that can be displayed in a scene graph.

13.3 Painter App: RadioButtons, Mouse Events and Shapes(Cont.)

- Mouse Event Handling

Mouse events	When the event occurs for a given node
`onMouseClicked`	When the user clicks a mouse button—that is, presses and releases a mouse button without moving the mouse—with the mouse cursor within that node.
`onMouseDragEntered`	When the mouse cursor enters a node's bounds during a mouse drag—that is, the user is moving the mouse with a mouse button pressed.
`onMouseDragExited`	When the mouse cursor exits the node's bounds during a mouse drag.
`onMouseDragged`	When the user begins a mouse drag with the mouse cursor within that node and continues moving the mouse with a mouse button pressed.

`onMouseDragOver`	When a drag operation that started in a *different* node continues with the mouse cursor over the given node.
`onMouseDragReleased`	When the user completes a drag operation that began in that node.
`onMouseEntered`	When the mouse cursor enters that node's bounds.
`onMouseExited`	When the mouse cursor exits that node's bounds.
`onMouseMoved`	When the mouse cursor moves within that node's bounds.
`onMousePressed`	When user presses a mouse button with the mouse cursor within that node's bounds.
`onMouseReleased`	When user releases a mouse button with the mouse cursor within that node's bounds.

```
// Fig. 13.5: Painter.java
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Painter extends Application {
   @Override
   public void start(Stage stage) throws Exception {
      Parent root =
         FXMLLoader.load(getClass().getResource("Painter.fxml"));

      Scene scene = new Scene(root);
      stage.setTitle("Painter"); // displayed in window's title bar
      stage.setScene(scene);
      stage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

```
3  // Fig. 13.6: PainterController.java
5  import javafx.event.ActionEvent;
6  import javafx.fxml.FXML;
7  import javafx.scene.control.RadioButton;
8  import javafx.scene.control.ToggleGroup;
9  import javafx.scene.input.MouseEvent;
10 import javafx.scene.layout.Pane;
11 import javafx.scene.paint.Color;
12 import javafx.scene.paint.Paint;
13 import javafx.scene.shape.Circle;
14
15 public class PainterController {
16    // enum representing pen sizes
17    private enum PenSize {
18       SMALL(2),
19       MEDIUM(4),
20       LARGE(6);
21
22       private final int radius;
23
24       PenSize(int radius) {this.radius = radius;}
25
26       public int getRadius() {return radius;}
27    };
28
```

```
29      // instance variables that refer to GUI components
30      @FXML private RadioButton blackRadioButton;
31      @FXML private RadioButton redRadioButton;
32      @FXML private RadioButton greenRadioButton;
33      @FXML private RadioButton blueRadioButton;
34      @FXML private RadioButton smallRadioButton;
35      @FXML private RadioButton mediumRadioButton;
36      @FXML private RadioButton largeRadioButton;
37      @FXML private Pane drawingAreaPane;
38      @FXML private ToggleGroup colorToggleGroup;
39      @FXML private ToggleGroup sizeToggleGroup;
40
41      // instance variables for managing Painter state
42      private PenSize radius = PenSize.MEDIUM; // radius of circle
43      private Paint brushColor = Color.BLACK; // drawing color
44
```

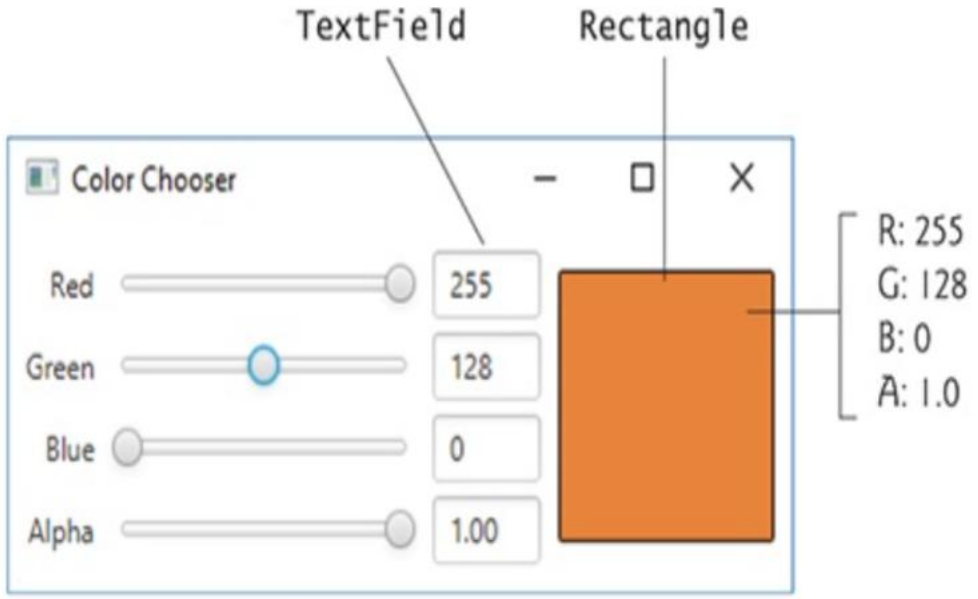
When the FXMLLoader creates a controller- class object, FXMLLoader determines whether the class contains an initialize method with no parameters and, if so, calls that method to initialize the controller.

```
45      // set user data for the RadioButtons
46      public void initialize() {
47          // user data on a control can be any Object
48          blackRadioButton.setUserData(Color.BLACK);
49          redRadioButton.setUserData(Color.RED);
50          greenRadioButton.setUserData(Color.GREEN);
51          blueRadioButton.setUserData(Color.BLUE);
52          smallRadioButton.setUserData(PenSize.SMALL);
53          mediumRadioButton.setUserData(PenSize.MEDIUM);
54          largeRadioButton.setUserData(PenSize.LARGE);
55      }
56
57      // handles drawingArea's onMouseDragged MouseEvent
58      @FXML
59      private void drawingAreaMouseDragged(MouseEvent e) {
60          Circle newCircle = new Circle(e.getX(), e.getY(),
61              radius.getRadius(), brushColor);
62          drawingAreaPane.getChildren().add(newCircle);
63      }
64
```

```
65     // handles color RadioButton's ActionEvents
66     @FXML
67     private void colorRadioButtonSelected(ActionEvent e) {
68        // user data for each color RadioButton is the corresponding Color
69        brushColor =
70           (Color) colorToggleGroup.getSelectedToggle().getUserData();
71     }
72
73     // handles size RadioButton's ActionEvents
74     @FXML
75     private void sizeRadioButtonSelected(ActionEvent e) {
76        // user data for each size RadioButton is the corresponding PenSize
77        radius =
78           (PenSize) sizeToggleGroup.getSelectedToggle().getUserData();
79     }
80
81     // handles Undo Button's ActionEvents
82     @FXML
83     private void undoButtonPressed(ActionEvent event) {
84        int count = drawingAreaPane.getChildren().size();
85
86        // if there are any shapes remove the last one added
87        if (count > 0) {
88           drawingAreaPane.getChildren().remove(count - 1);
89        }
90     }
91
92     // handles Clear Button's ActionEvents
93     @FXML
94     private void clearButtonPressed(ActionEvent event) {
95        drawingAreaPane.getChildren().clear(); // clear the canvas
96     }
97  }
```

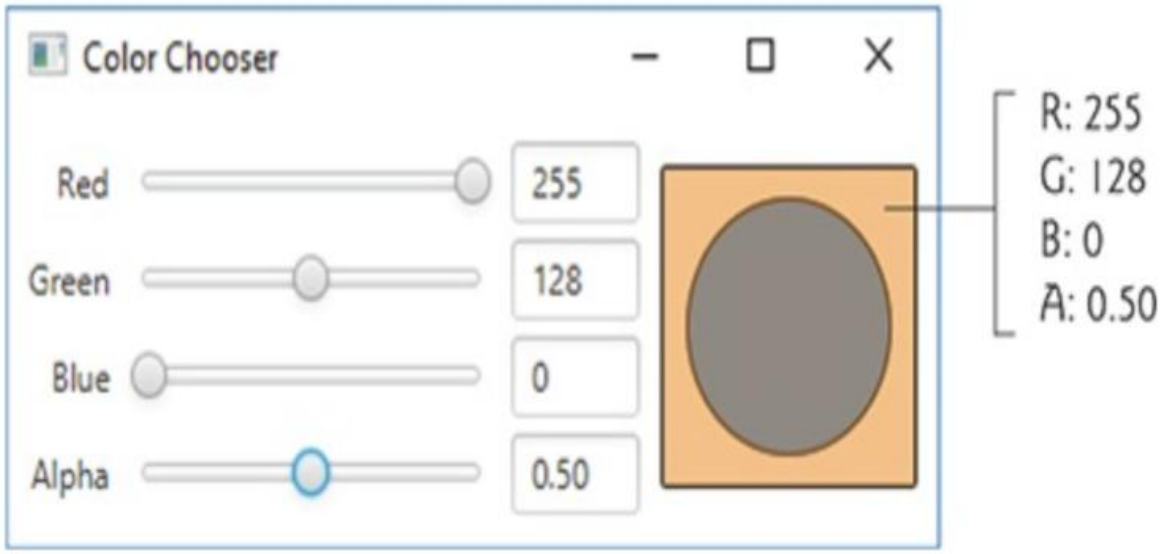
13.4 Color Chooser App: Property Bindings and Property Listeners

a) Using the **Red** and **Green** `Sliders` to create an opaque orange color



13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

b) Using the **Red**, **Green** and **Alpha** Sliders to create a semitransparent orange color—notice that the semitransparent orange mixes with the color of the circle behind the colored square



13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

- RGBA color system
  - every color is represented by its red, green and blue color values, each ranging from 0 to 255, where 0 denotes no color and 255 full color.
  - The alpha value (A)—which ranges from 0.0 to 1.0—represents a color's opacity, with 0.0 being completely transparent and 1.0 completely opaque

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

A property is defined by creating set and get methods with specific naming conventions.

Software Engineering Observation 13.2

Methods that define properties should be declared `final` *to prevent subclasses from overriding the methods, which could lead to unexpected results in client code.*

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

- Property binding
  - JavaFX properties are implemented in a manner that makes them observable—when a property's value changes, other objects can respond accordingly.
  - Package javafx.beans.property contains many classes that you can use to define bindable properties in your own classes.

```
redTextField.textProperty().bind(
   redSlider.valueProperty().asString("%.0f"));
```

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

- Property Listeners
  - A property listener is an event handler that's invoked when a property's value changes.

```
redSlider.valueProperty().addListener(
    new ChangeListener<Number>() {
       @Override
    public void changed(ObservableValue<? extends Number> ov,  Number oldValue,
Number newValue) {
       red=newValue.intValue();
       colorRectangle.setFill(Color.rgb(red, green, blue, alpha));
           }
       }
    );
```

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

```
// Fig. 13.8: ColorChooser.java
import javafx.application.Application;

public class ColorChooser extends Application {
   @Override
   public void start(Stage stage) throws Exception {
      Parent root =
         FXMLLoader.load(getClass().getResource("ColorChooser.fxml"));

      Scene scene = new Scene(root);
      stage.setTitle("Color Chooser");
      stage.setScene(scene);
      stage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

```
 5 import javafx.beans.value.ChangeListener;
12
13 public class ColorChooserController {
14    // instance variables for interacting with GUI components
15    @FXML private Slider redSlider;
16    @FXML private Slider greenSlider;
17    @FXML private Slider blueSlider;
18    @FXML private Slider alphaSlider;
19    @FXML private TextField redTextField;
20    @FXML private TextField greenTextField;
21    @FXML private TextField blueTextField;
22    @FXML private TextField alphaTextField;
23    @FXML private Rectangle colorRectangle;
24
25    // instance variables for managing
26    private int red = 0;
27    private int green = 0;
28    private int blue = 0;
29    private double alpha = 1.0;
30
31    public void initialize() {
32       // bind TextField values to corresponding Slider values
33       redTextField.textProperty().bind(
34          redSlider.valueProperty().asString("%.0f"));
35       greenTextField.textProperty().bind(
36          greenSlider.valueProperty().asString("%.0f"));
37       blueTextField.textProperty().bind(
38          blueSlider.valueProperty().asString("%.0f"));
39       alphaTextField.textProperty().bind(
40          alphaSlider.valueProperty().asString("%.2f"));
41
```

13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

```
42        // listeners that set Rectangle's fill based on Slider changes
43
44        redSlider.valueProperty().addListener(
45           new ChangeListener<Number>() {
46              @Override
47              public void changed(ObservableValue<? extends Number> ov,
48                 Number oldValue, Number newValue) {
49                 red = newValue.intValue();
50                 colorRectangle.setFill(Color.rgb(red, green, blue, alpha));
51              }
52           }
53        );
54        greenSlider.valueProperty().addListener(
55           new ChangeListener<Number>() {
56              @Override
57              public void changed(ObservableValue<? extends Number> ov,
58                 Number oldValue, Number newValue) {
59                 green = newValue.intValue();
60                 colorRectangle.setFill(Color.rgb(red, green, blue, alpha));
61              }
62           }
63        );
```

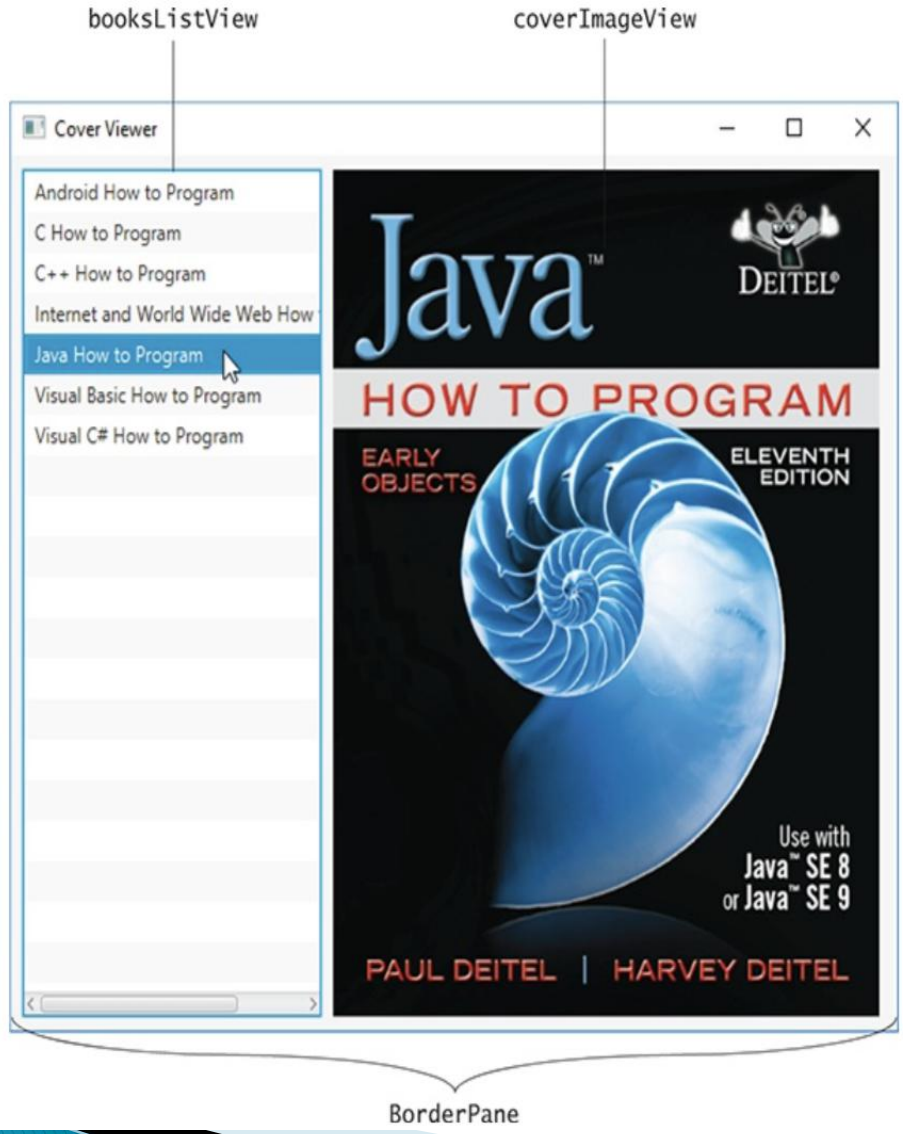
13.4 Color Chooser App: Property Bindings and Property Listeners(Cont.)

```
64          blueSlider.valueProperty().addListener(
65             new ChangeListener<Number>() {
66                @Override
67                public void changed(ObservableValue<? extends Number> ov,
68                   Number oldValue, Number newValue) {
69                   blue = newValue.intValue();
70                   colorRectangle.setFill(Color.rgb(red, green, blue, alpha));
71                }
72             }
73          );
74          alphaSlider.valueProperty().addListener(
75             new ChangeListener<Number>() {
76                @Override
77                public void changed(ObservableValue<? extends Number> ov,
78                   Number oldValue, Number newValue) {
79                   alpha = newValue.doubleValue();
80                   colorRectangle.setFill(Color.rgb(red, green, blue, alpha));
81                }
82             }
83          );
84       }
85    }
```

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.

13.5 Cover Viewer App: Data-Driven GUIs with JavaFX Collections(Cont.)



13.5 Cover Viewer App: Data-Driven GUIs with JavaFX Collections(Cont.)

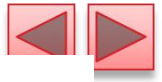
- For the ListView, set the following properties:
  - Margin—8 (for the right margin) to separate the ListView from the ImageView
  - Pref Width—200
  - Max Height—MAX_VALUE
  - Min Width, Min Height, Pref Height and Max Width —USE_COMPUTED_SIZE

13.5 Cover Viewer App: Data-Driven GUIs with JavaFX Collections(Cont.)

- For the ImageView, set the following properties:
  - Fit Width and Fit Height — 370 and 480
- For the BorderPane, set the following properties:
  - Pref Width and Pref Height —USE_COMPUTED_SIZE
  - Padding —8

```
3  // Fig. 13.13: CoverViewer.java
5  import javafx.application.Application;
10
11 public class CoverViewer extends Application {
12    @Override
13    public void start(Stage stage) throws Exception {
14       Parent root =
15          FXMLLoader.load(getClass().getResource("CoverViewer.fxml"));
16
17       Scene scene = new Scene(root);
18       stage.setTitle("Cover Viewer");
19       stage.setScene(scene);
20       stage.show();
21    }
22
23    public static void main(String[] args) {
24       launch(args);
25    }
26 }
27
```

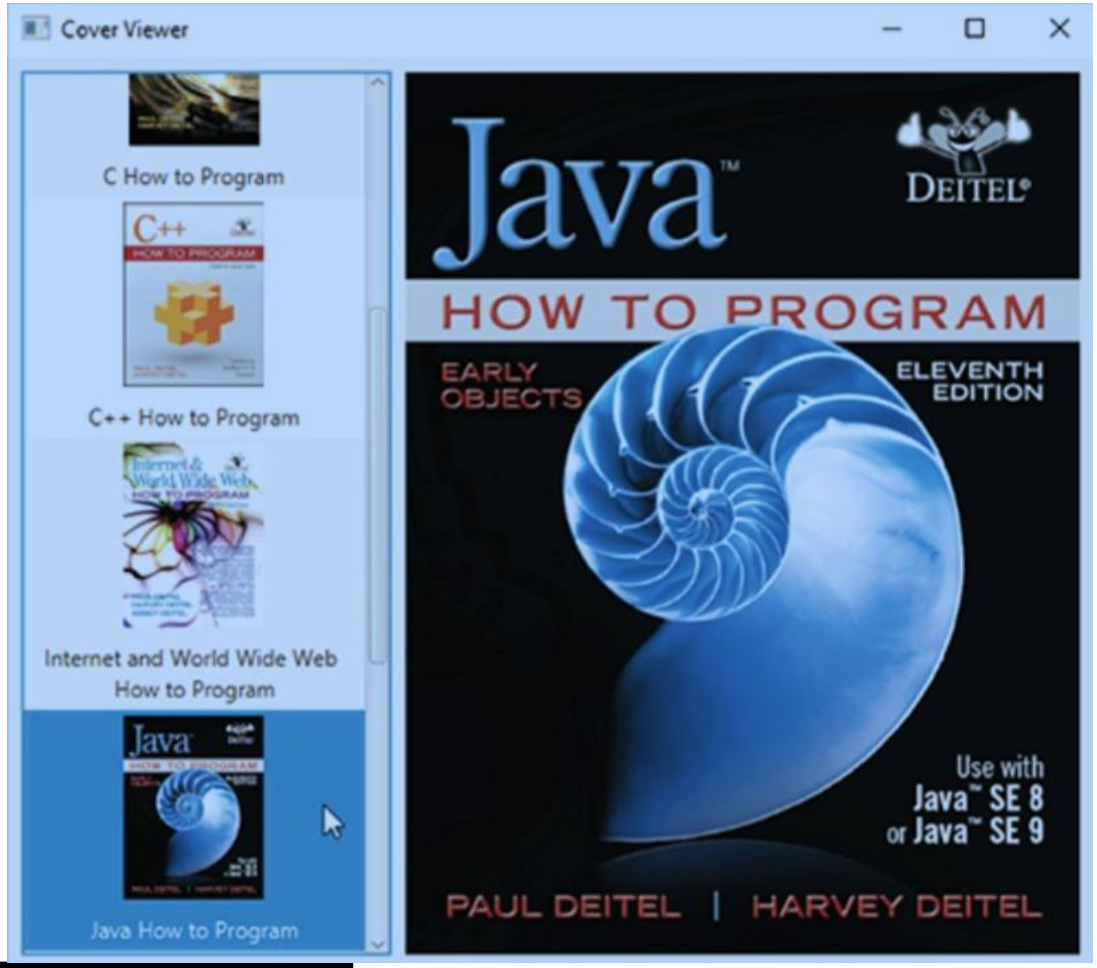
```java
package ch13.CoverViewer;

// Book.java
public class Book {
   private String title; // book title
   private String thumbImage; // source of book cover's thumbnail image
   private String largeImage; // source of book cover's full-size image

   public Book(String title, String thumbImage, String largeImage) {
      this.title = title;
      this.thumbImage = thumbImage;
      this.largeImage = largeImage;
   }

   public String getTitle() {return title;}

   public void setTitle(String title) {this.title = title;}

   public String getThumbImage() {return thumbImage;}

   public void setThumbImage(String thumbImage) {this.thumbImage = thumbImage;}

   public String getLargeImage() {return largeImage;}

   public void setLargeImage(String largeImage) {this.largeImage = largeImage;}

   @Override
   public String toString() {return getTitle();}
}

```



```
3  // Fig. 13.14: CoverViewerController.java
5  import javafx.beans.value.ChangeListener;
6  import javafx.beans.value.ObservableValue;
7  import javafx.collections.FXCollections;
8  import javafx.collections.ObservableList;
9  import javafx.fxml.FXML;
10 import javafx.scene.control.ListView;
11 import javafx.scene.image.Image;
12 import javafx.scene.image.ImageView;
13
14 public class CoverViewerController {
15    // instance variables for interacting with GUI
16    @FXML private ListView<Book> booksListView;
17    @FXML private ImageView coverImageView;
18
19    // stores the list of Book Objects
20    private final ObservableList<Book> books =
21       FXCollections.observableArrayList();
22
23    // initialize controller
24    public void initialize() {
25       // populate the ObservableList<Book>
26       books.add(new Book("Android How to Program",
27              this.getClass().getResource("images/small/androidhtp.jpg").getPath(),
28              this.getClass().getResource("images/large/androidhtp.jpg").getPath()));
29       books.add(new Book("C How to Program",
30              this.getClass().getResource("images/small/chtp.jpg").getPath(),
31              this.getClass().getResource("images/large/chtp.jpg").getPath()));
32       books.add(new Book("C++ How to Program",
33              this.getClass().getResource("images/small/cpphtp.jpg").getPath(),
34              this.getClass().getResource( "images/large/cpphtp.jpg").getPath()));
35       books.add(new Book("Internet and World Wide Web How to Program",
36              this.getClass().getResource("images/small/iw3htp.jpg").getPath(),
37              this.getClass().getResource("images/large/iw3htp.jpg").getPath()));
```

```
38          books.add(new Book("Java How to Program",
39                  this.getClass().getResource("images/small/jhtp.jpg").getPath(),
40                  this.getClass().getResource("images/large/jhtp.jpg").getPath()));
41          books.add(new Book("Visual Basic How to Program",
42                  this.getClass().getResource("images/small/vbhtp.jpg").getPath(),
43                  this.getClass().getResource("images/large/vbhtp.jpg").getPath()));
44          books.add(new Book("Visual C# How to Program",
45                  this.getClass().getResource("images/small/vcshtp.jpg").getPath(),
46                  this.getClass().getResource( "images/large/vcshtp.jpg").getPath()));
47          booksListView.setItems(books); // bind booksListView to books
48
49          // when ListView selection changes, show large cover in ImageView
50          booksListView.getSelectionModel().selectedItemProperty().
51             addListener(
52                new ChangeListener<Book>() {
53                   @Override
54                   public void changed(ObservableValue<? extends Book> ov,
55                      Book oldValue, Book newValue) {
56                      coverImageView.setImage(
57                              new Image("file:" + newValue.getLargeImage()));
58
59                   }
60                }
61             );
62       }
63
64    }
```

13.6 Cover Viewer App: Customizing ListView Cells



13.6 Cover Viewer App: Customizing ListView Cells

- 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.
- Use the ListView's setCellFactory method to replace the default cell factory with one that returns objects of the ListCell subclass.

13.6 Cover Viewer App: Customizing ListView Cells

- Programmatically Creating Layouts and Controls
  - 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.

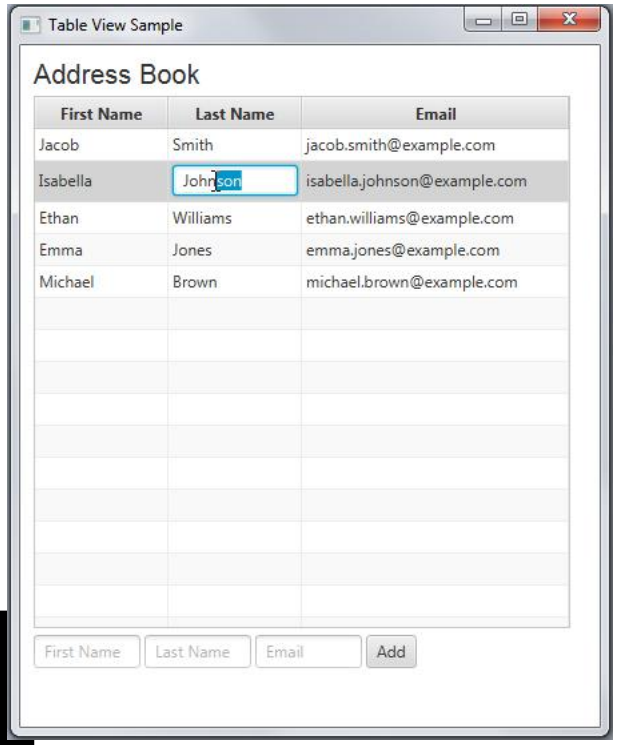
```
import javafx.geometry.Pos;

public class ImageTextCell extends ListCell<Book> {
   private VBox vbox = new VBox(8.0); // 8 points of gap between controls
   private ImageView thumbImageView = new ImageView(); // initially empty
   private Label label = new Label();

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

      thumbImageView.setPreserveRatio(true);
      thumbImageView.setFitHeight(100.0); // thumbnail 100 points tall
      vbox.getChildren().add(thumbImageView); // attach to Vbox

      label.setWrapText(true); // wrap if text too wide to fit in label
      label.setTextAlignment(TextAlignment.CENTER); // center text
      vbox.getChildren().add(label); // attach to VBox

      setPrefWidth(USE_PREF_SIZE); // use preferred size for cell width
   }

   // called to configure each custom ListView cell
   @Override
   protected void updateItem(Book item, boolean empty) {
      // required to ensure that cell displays properly
      super.updateItem(item, empty);

      if (empty || item == null) {
         setGraphic(null); // don't display anything
      }
      else {
         // set ImageView's thumbnail image
         thumbImageView.setImage(new Image("file:" + item.getThumbImage()));
         label.setText(item.getTitle()); // configure Label's text
         setGraphic(vbox); // attach custom layout to ListView cell
      }
   }
}
```

```
16 public class CoverViewerController {
17    // instance variables for interacting with GUI
18    @FXML private ListView<Book> booksListView;
19    @FXML private ImageView coverImageView;
20
21    // stores the list of Book Objects
22    private final ObservableList<Book> books =
23       FXCollections.observableArrayList();
24
25    public void initialize() {
26       // populate the ObservableList<Book>
27       books.add(new Book("Android How to Program",
28          this.getClass().getResource("images/small/androidhtp.jpg").getFile(), thi
29       books.add(new Book("C How to Program",
30             this.getClass().getResource("images/small/chtp.jpg").getFile(), this
31       books.add(new Book("C++ How to Program",
32             this.getClass().getResource("images/small/cpphtp.jpg").getFile(), th
33       books.add(new Book("Internet and World Wide Web How to Program",
34             this.getClass().getResource("images/small/iw3htp.jpg").getFile(),  t
35       books.add(new Book("Java How to Program",
36             this.getClass().getResource( "images/small/jhtp.jpg").getFile(), thi
37       books.add(new Book("Visual Basic How to Program",
38             this.getClass().getResource("images/small/vbhtp.jpg").getFile(), thi
39       books.add(new Book("Visual C# How to Program",
40             this.getClass().getResource("images/small/vcshtp.jpg").getFile(), th
41       booksListView.setItems(books); // bind booksListView to books
42
```

```
43          // when ListView selection changes, show large cover in ImageView
44          booksListView.getSelectionModel().selectedItemProperty().
45             addListener(
46                new ChangeListener<Book>() {
47                   @Override
48                   public void changed(ObservableValue<? extends Book> ov,
49                      Book oldValue, Book newValue) {
50                      coverImageView.setImage(
51                         new Image("file:" + newValue.getLargeImage()));
52                   }
53                }
54             );
55
56          // set custom ListView cell factory
57          booksListView.setCellFactory(
58             new Callback<ListView<Book>, ListCell<Book>>() {
59                @Override
60                public ListCell<Book> call(ListView<Book> listView) {
61                   return new ImageTextCell();
62                }
63             }
64          );
65       }
66    }
67
```

13.7 Additional JavaFX Capabilities

- TableView
  - JavaFX's TableView control (package javafx.scene.control) displays tabular data in rows and columns, and supports user interactions with that data.



Example: http://www.javafxchina.net/blog/2015/04/doc03_tableview/

13.7 Additional JavaFX Capabilities(Cont.)

- Accessibility
  - In a Java SE 8 update, JavaFX added accessibility features to help people with visual impairments use their devices.

13.7 Additional JavaFX Capabilities(Cont.)

- Third-Party JavaFX Libraries
  - ControlsFX
    - http://www.controlsfx.org/
  - JFXtras
    - http://jfxtras.org/
  - Medusa
    - https://github.com/HanSolo/Medusa/blob/master/README.md.

13.7 Additional JavaFX Capabilities(Cont.)

- Creating Custom JavaFX Controls
  - We can create custom controls by extending existing JavaFX control classes to customize them or by extending JavaFX's Control class directly.

13.8 Multiple Stages

- Project TwoStages
- Project TwoControllers