1 **JavaFX GUI's (Part 2)**

CST242

2 **JavaFX Event Handlers (Review)**

- There are three ways to create event handlers in JavaFX:
  1. A nested class (or possibly a separate class file) that implements a Java API event handler interface
  2. Anonymous inner classes
  3. Lambda expression event handling

3 **The EventHandler Interface (Page 1)**

- EventHandler is a Java API interface used to manage *event listening* and *event handling* for JavaFX GUI nodes
- Objects instantiated from a class that implements EventHandler interface "are" event handlers
- Imported from javafx.event package:
  import javafx.event.EventHandler;

4 **The EventHandler Interface (Page 2)**

- Format:
  private class *EventHandlerClassName* implements EventHandler<ActionEvent>
  { …
  - implements rather than extends
  - ActionEvent is the type for Buttons, RadioButtons, CheckBoxes, etc.
- Example:
  private class GreetingEventHandler implements EventHandler<ActionEvent>
  { …

5 **The handle Method (Page 1)**

- Event handler classes have a method handle from the abstract method from the interface it implements
- The method automatically is called when the event associated with the event handler is received by the node that registered the handler

6 **The handle Method (Page 2)**

- The parameter variable "event" provides access to ActionEvent methods and properties
- Example:

```
private class GreetingEventHandler implements EventHandler<ActionEvent>
{
    @Override
    public void handle(ActionEvent event)
    {
        System.out.println("OK clicked");
    }
```

```
}
```

7 **Instantiating an EventHandler Object**

- An EventHandler *class* must be defined previously
- Format:

  *EventHandlerClass eventHandlerObject* = new *EventHandlerConstructor*();
- Example:

  GreetingEventHandler eventHandler = new GreetingEventHandler();

8 **The setOnAction Method**

- Method that assigns an EventHandler object to the node
- This method effectively *activates* event listening
- Format:

  *NodeClass nodeObject* = new *Constructor*(...);

  *nodeObject*.setOnAction(*eventHandlerObject*);
- Example:

  Button buttonHello = new Button("Hello");

  buttonHello.setOnAction(eventHandler);

10 **Anonymous Inner Classes (Page 1)**

- An anonymous inner class is an *event handler* without a name
  - Located inside the definition of application window in start method
  - Defined within the setOnAction method
- Combines creating object with defining of the event handler class

11 **Anonymous Inner Classes (Page 2)**

- Format:

```
NodeClass nodeObject = new Constructor(...);
nodeObject.setOnAction(new EventHandler<ActionEvent>()
{
   @Override
   public void handle(ActionEvent event)
   {
      statements
   }
}
);
```

12 **Anonymous Inner Classes (Page 3)**

- Example:

```
Button buttonHello = new Button("Hello");
buttonHello.setOnAction(new EventHandler<ActionEvent>()
{
   @Override
```

```
        public void handle(ActionEvent event)
        {
            System.out.println("Hello");
        }
    }
    );
```

14 **Lambda Expression Event Handling (Page 1)**

- Lambda expression event handling is a new feature in Java 8 which *replaces* anonymous inner class with a *more concise syntax*
- Also defined within the setOnAction() method for a Button or other node where a single *method* replaces the inner class
- A parameter (e) and the lambda operator (->) point to the method

15 **Lambda Expression Event Handling (Page 2)**

- Format:

```
NodeClass nodeObject = new Constructor( ... );
nodeObject.setOnAction( (e) ->
{
    statements
}
);
```

- The parameter variable e (or other programmer-defined variables) may be explicitly declared by type or the type inferred by the compiler)

16 **Lambda Expression Event Handling (Page 3)**

- Example 1 (execute one or more statements):

```
Button buttonHello = new Button("Hello");
buttonHello.setOnAction( (e) ->
{
    System.out.println("Hello");
    System.out.println("How are you?");
}
);
```

17 **Lambda Expression Event Handling (Page 4)**

- Example 2 (call a method):

```
Button buttonHello = new Button("Hello");
buttonHello.setOnAction( (e) ->
{
    displayHelloMessage();
}
);
```

18 **Lambda Expression Event Handling (Page 5)**

- The lambda expression may point directly to a *method call*
- Parameter variable e does not have to be wrapped inside (parentheses)
- Format:

  *NodeClassName object* = new *ConstructorName*( ... );
  *object*.setOnAction( e -> *methodCall*( *parameters* ... ) );

19 **Lambda Expression Event Handling (Page 6)**

- Examples:

```
Button buttonOK = new Button("OK");
buttonOK.setOnAction( e -> JOptionPane.showMessageDialog(null, "OK clicked") );

Button buttonHello = new Button("Hello");
buttonHello.setOnAction( e -> handleClick(e) );
```

20 **The setPrefWidth Method**

- Sets the value of the property prefWidth for a JavaFX control
- Should be set only if the control's internally computed preferred (default) width does not meet the application's layout needs
- Format:

  *nodeObject*.setPrefWidth(*widthValue*);
- Example:

```
labelHeader.setPrefWidth(580);
```

21 **The setMinWidth Method**

- Sets the value of the property minWidth for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:

  *nodeObject*.setMinWidth(*widthValue*);
- Example:

```
labelHeader.setMinWidth(450);
```

22 **The setMaxWidth Method**

- Sets the value of the property maxWidth for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:

  *nodeObject*.setMaxWidth(*widthValue*);
- Example:

```
labelHeader.setMaxWidth(580);
```

23 **The setPrefHeight Method**

- Sets the value of the property prefHeight for a JavaFX control
- Should be set only if the control's internally computed preferred (default) width does not meet the application's layout needs

- Format:
  *nodeObject*.setPrefHeight(*heightValue*);
- Example:
  buttonHello.setPrefHeight(60);

24 **The setMinHeight Method**

- Sets the value of the property minHeight for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:
  *nodeObject*.setMinHeight(*heightValue*);
- Example:
  buttonHello.setMinWidth(50);

25 **The setMaxHeight Method**

- Sets the value of the property maxWidth for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:
  *nodeObject*.setMaxHeight(*heightValue*);
- Example:
  buttonHello.setMaxHeight(60);

26 **The setPrefSize Method**

- Sets the values of the properties prefWidth and prefHeight for a JavaFX control
- Should be set only if the control's internally computed preferred (default) width does not meet the application's layout needs
- Format:
  *nodeObject*.setPrefSize(*widthValue*, *heightValue*);
- Example:
  buttonHello.setPrefSize(100, 60);

27 **The setMinSize Method**

- Sets the values of the properties minWidth and minHeight for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:
  *nodeObject*.setMinSize(*widthValue*, *heightValue*);
- Example:
  buttonHello.setMinSize(80, 50);

28 **The setMaxSize Method**

- Sets the values of the properties maxWidth and maxHeight for a JavaFX control
- Should be set if the app likely will run on displays of varying sizes
- Format:
  *nodeObject*.setMaxSize(*widthValue*, *heightValue*);
- Example:

```
buttonHello.setMaxSize(100, 60);
```

30 **The Font Class (Page 1)**

- Creates a Font object that sets font properties of a JavaFX object:
  - Typeface (font name)
  - Bold property
  - Posture (italic) property
  - Font size
- Located in the javafx.scene.text package:

```
import javafx.scene.text.Font;
```

31 **The Font Class (Page 2)**

- Format to declare a Font object:

  Font *fontObject* = new Font( [*typeFace*,] *size*);
  - *typeFace* is a String which is the "name" of a typeface (a.k.a. font) (optional)
  - *size* is numeric value of type double which is font size measured in "points"
- Examples:

```
Font font = new Font(28);
Font font = new Font("Consolas", 28);
```

32 **The setFont Method**

- Sets font for a JavaFX node object (Label, TextField, Button, RadioButton, CheckBox, etc.)
- Format:

  *nodeObject*.setFont(*fontObject*);
- Examples:

```
labelTitle.setFont( new Font(28) );
labelTitle.setFont( new Font("Consolas", 28) );
```

33 **The Color Class (Page 1)**

- Creates a Color object that uses the RGB model
  - Values are between zero (0%) to 1.0 (100%)
  - Represents amount of red, green and blue in makeup of the color)
- Located in the javafx.scene.paint package:

```
import javafx.scene.paint.Color;
```

34 **The Color Class (Page 2)**

- Format to declare a Color object:

  Color *colorObject* = new Color( *red*, *green*, *blue*, [*opacity*] );
- Examples:

```
Color color = new Color(1.0, 0.0, 0.0);
labelTitle.setTextFill(color);
=======================================
labelTitle.setTextFill( new Color(1.0, 1.0, 0.0) );
```

35 **The Color Constants**

- A series of over 200 standard static constants of class Color that return a hexadecimal value representing an RGB color
- A few examples of these constants are BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE and YELLOW
- Example:

```
labelTitle.setTextFill(Color.PEACHPUFF);
```

36 **The Color.web method (Page 1)**

- The static method web from class Color returns a color as a hexadecimal RGB color value
- The *color* argument may be one of the following:
  - A valid "color string name" from the Color class
  - The hexadecimal RGB color value in which the first two digits are the amount of red, the next two digits the amount of green and the last two digits the amount of blue, e.g. #683200 (value range from 00 to FF)

37 **The Color.web method (Page 2)**

- Format:

  Color.web(*color*)
  - The *color* is either "color name string" or a hexadecimal RGB color value
- Examples:

```
labelTitle.setTextFill( Color.web("orange") );
labelTitle.setTextFill( Color.web("#683200") );
```

38 **The setTextFill Method**

- Sets the text color for a JavaFX node object (Label, TextField, Button, RadioButton, CheckBox, etc.)
- Format:

  *nodeObject*.setTextFill(*color*);
- Examples:

```
labelTitle.setTextFill(Color.ORANGE);
labelTitle.setTextFill( Color.web("orange") );
labelTitle.setTextFill( Color.web("#683200") );
labelTitle.setTextFill( new Color(1.0, 1.0, 0.0) );
```

40 **The RadioButton Class (Page 1)**

- A RadioButton is a clickable node designed to turn "on" and "off"
- Normal functionality is that when one button is "group" is clicked "on", another button in group automatically turns "off"
  - Cannot be clicked to turn it "off"
- Located in the javafx.scene.control package:

```
import javafx.scene.control.RadioButton;
```

41 **The RadioButton Class (Page 2)**

- Format to instantiate:
  RadioButton *radioButtonObject* = new RadioButton( [*textString*] );
  - *textString* is descriptive label that appears to right of the radio button (optional)
- Examples:
  RadioButton radioButtonBase = new RadioButton("Base Model");

42 **The ToggleGroup Class (Page 1)**

- ToggleGroup is a class that references a group of variables managed so that only a single "Toggle" (e.g. RadioButton) within the group may be selected at any one time
- Located in the javafx.scene.control package:
  import javafx.scene.control.ToggleGroup;

43 **The ToggleGroup Class (Page 2)**

- Format to instantiate:
  ToggleGroup *toggleGroupObject* = new ToggleGroup();
  - Constructor takes no parameters
- Example:
  ToggleGroup toggleGroup = new ToggleGroup();

44 **The getToggles Method**

- For a ToggleGroup object, returns all "Toggles" (e.g. RadioButtons) in the group
- Format:
  *toggleGroupObject*.getToggles()
- Example:
  toggleGroup.getToggles().addAll(radioButtonBase, radioButtonSport,
  radioButtonDeluxe);

45 **The addAll Method for a ToggleGroup**

- The addAll method for returned list of "Toggles" adds toggles or additional toggles to the group
- Format:
  *toggleGroupObject*.getToggles().addAll( *toggles* )
  - *toggles* could be a list of RadioButtons
- Example:
  toggleGroup.getToggles().addAll(radioButtonBase, radioButtonSport,
  radioButtonDeluxe);

46 **The setSelected Method**

- Programatically turns "on" or "off" a RadioButton
  - Also used for a CheckBox
- May be set "on" for *only one* RadioButton in the ToggleGroup at any given moment
- Format:
  *radioButtonObject*.setSelected( true | false );

- Example:
  radioButtonBase.setSelected(true);

47 **The isSelected Method**

- A boolean method that returns true or false value for a RadioButton to indicate if it is "on" or "off"
  - Also used for a CheckBox
- Format:
  *radioButtonObject*.isSelected()
- Example:
  if ( radioButtonBase.isSelected() )
  { …

48 **The VBox Class (Page 1)**

- Layout control that is a "pane" that places nodes in a single column
  - Unlimited number of columns as needed
- Located in the javafx.scene.layout package
  import javafx.scene.layout.VBox;

49 **The VBox Class (Page 2)**

- Format to instantiate and add nodes simultaneously:
  VBox *vBoxObject* = new VBox( [*spacing*,] [*children*] );
  - *spacing* is amount of spacing between rows measured in pixels
  - *children* is a *list* of the initial nodes added to VBox
- Example:
  VBox vBoxRadio = new VBox(20, radioButtonBase, radioButtonSport,
  radioButtonDeluxe);

50 **The HBox Class (Page 1)**

- Layout control that is a "pane" that places nodes in a single row
  - Unlimited number of rows as needed
- Located in the javafx.scene.layout package
  import javafx.scene.layout.HBox;

51 **The HBox Class (Page 2)**

- Format to instantiate:
  HBox *hBoxObject* = new HBox(20, *children*);
  - *spacing* is amount of spacing between columns measured in pixels
  - *children* is a *list* of the initial nodes added to HBox
- Example:
  HBox hBoxRadio = new HBox(20, radioButtonBase, radioButtonSport,
  radioButtonDeluxe);

52 **The Insets Class (Page 1)**

- Insets is a JavaFX class used to create *padding* between outer edges of a layout object

("box" in which it is contained) and its contents
- Located in the javafx.geometry package
  import javafx.geometry.Insets;

53 **The Insets Class (Page 2)**

- Formats to instantiate:
  Insets *insetsObject* = new Insets( *topRightBottomLeft* );
  Insets *insetsObject* = new Insets(*top*, *right*, *bottom*, *left*);
  - *topRightBottomLeft* is single argument that sets padding equally for all sides
- Examples:
  Insets insets = new Insets(10);
  Insets insets = new Insets(20, 10, 20, 10);

56 **The CheckBox Class (Page 1)**

- A CheckBox is a clickable node designed to turn "on" and "off"
- The normal functionality is that the first time the box is clicked it is turned "on" and the next time it is clicked it is turned "off"
  - CheckBoxes are *not grouped*
- Located in the javafx.scene.control package:
  import javafx.scene.control.CheckBox;

57 **The CheckBox Class (Page 2)**

- Format to instantiate:
  CheckBox *checkBoxObject* = new CheckBox( [*textString*] );
  - *textString* is descriptive label that appears to right of the check box (optional)
- Examples:
  CheckBox checkBoxExtend = new CheckBox("Extended Warranty");

58 **The GridPane Class (Review)**

- Layout pane that allows the developer to create a flexible grid of row and columns to layout nodes
- Nodes can be placed in any cell as needed
  - The GridPane automatically grows horizontally and/or vertically as new cells (or rows or columns) are added
- Located in the javafx.scene.layout package:
  import javafx.scene.layout.GridPane;

59 **The addRow Method**

- Adds *all nodes* to a row in a GridView control in a single statement
- Format:
  grid.addRow(*rowNumber*, *children*);
  - *rowNumber* where the row will be inserted in which the first row is row number zero (0)
  - *children* is a *list* of the nodes inserted into the row

- Example:
  grid.addRow(1, vBoxRadio, vBoxCheck, vBoxResults);

60 **The addColumn Method**

- Adds *all nodes* to a column in a GridView control in a single statement
- Format:
  grid.addColumn(*columnNumber*, *children*);
  - *columnNumber* where the column will be inserted in which the first column is column number zero (0)
  - *children* is a *list* of the nodes inserted into the column
- Example:
  grid.addColumn(2, vBoxRadio, vBoxCheck, vBoxResults);

61 **The setColumnSpan Method (Page 1)**

- The method setColumnSpan is a static method that horizontally *merges* more than one cell of a GridPane into a single cell
- May be set to the constant GridPane.REMAINING, which will cause the span to extend across all the remaining columns

62 **The setColumnSpan Method (Page 2)**

- Format:
  GridPane.setColumnSpan(*nodeObject*, *numberOfColumns*);
  - *The nodeObject* must have previously been inserted into the GridPane using either add or addRow
- Example:
  grid.addRow(0, labelTitle);
  GridPane.setColumnSpan(labelTitle, 3);

63 **The setRowSpan Method (Page 1)**

- The method setRowSpan is a static method that vertically *merges* more than one cell of a GridPane into a single cell
- May be set to the constant GridPane.REMAINING, which will cause the span to extend across all the remaining rows

64 **The setRowSpan Method (Page 2)**

- Format:
  GridPane.setRowSpan(*nodeObject*, *numberOfRows*);
  - *The nodeObject* must have previously been inserted into the GridPane using either add or addColumn
- Example:
  grid.addColumn(0, labelTitle);
  GridPane.setRowSpan(labelTitle, GridPane.REMAINING);

67 **The TitledPane Class (Page 1)**

- A TitledPane is a layout panel with a title that can be opened and closed ("drill down")

- Only title remains displayed when the panel closes
- A single node or a nested layout object may be inserted into it
  - A VBox often is inserted since TitledPane opens in a downward direction
- Located in the javafx.scene.control package:
  import javafx.scene.control.TitledPane;

68 **The TitledPane Class (Page 2)**

- Format to instantiate:
  TitledPane *titledPaneObject* = new TitledPane("*title*", *contentNode*);
  - *title* is displayed at top of the pane and remains displayed when it closes
  - *contentNode* is another node or a layout object
- Example:
  TitledPane titledPaneRadio = new TitledPane("Model", vBoxRadio);

69 **The setStyle Method (Page 1)**

- The setStyle method can be applied to many JavaFX nodes and applies CSS (Cascading Style Sheets) formatting
- Similar to formatting applied in HTML web pages

70 **The setStyle Method (Page 2)**

- Each style element is applied in a *property* and *value* format (the property name begins "-fx-"):
  -fx-*property*: *value*;
- This process called "skinning" provides a very powerful way of formatting JavaFX elements

71 **The setStyle Method (Page 3)**

- Format:
  *javaFXObject*.setStyle("-fx-*property1*: *value1*;
  -fx-*property2*: *value2*; ... ");
- Example:
  labelHeader.setStyle("-fx-font-size: 22pt;
  -fx-background-color: rgb(104, 50, 0);
  -fx-text-fill: white;");

72 **The -fx-font-family Property**

- Specifies the typeface (a.k.a. font) for the text for a JavaFX node
- Format:
  -fx-font-family: '*typeface names*';
  - The *typeface names* are a *list* of typefaces from which the operating system selects the first from the list that is installed
  - Any '*typeface name*' more than one word is entered within single quotes
- Example:
  labelHeader.setStyle("-fx-font-size: 'Comic Sans MS';");

73 **The -fx-font-size Property**

- Specifies font size for the text for a JavaFX node
- Format:

  -fx-font-size: *length* | medium | xx-small | x-small | small |large | x-large | xx-large | smaller | larger;

  - *length* measured in points (pt), pixels (px), etc.
- Example:

  labelHeader.setStyle("-fx-font-size: 22pt;");

74 **The -fx-text-fill Property**

- Specifies color for the text for a JavaFX node
- Format:

  -fx-text-fill: *color* | rgb(*red*, *green*, *blue*);

  - *color* is one of over 200 color names, e.g. white, red, blue, dark-blue, etc.
  - The rgb function specifies RGB color values between zero (0) and 255
- Example:

  labelHeader.setStyle("-fx-text-fill: white;");

75 **The -fx-background-color Property**

- Specifies background color behind a JavaFX node
- Format:

  -fx-background-color: *color* | rgb(*red*, *green*, *blue*);

  - The *color* is one of over 200 color names, e.g. white, red, blue, dark-blue, etc.
  - The rgb function specifies RGB color values between zero (0) and 255
- Example:

  labelHeader.setStyle("-fx-background-color: rgb(104, 50, 0);");

76 **The -fx-font-weight Property**

- Specifies bold formatting for a JavaFX node
- Format:

  -fx-font-weight: normal | bold | bolder | lighter | *number*;

  - *number* is a shade of bold, e.g. 100, 200, … 900
- Example:

  buttonCalculate.setStyle("-fx-font-weight: bold;");

77 **The -fx-border-color Property**

- Specifies border color around a JavaFX node
- Format:

  -fx-border-color: *color* | rgb(*red*, *green*, *blue*);

  - *color* is one of over 200 color names, e.g. white, red, blue, dark-blue, etc.
  - The rgb function specifies RGB color values between zero (0) and 255
- Example:

  vBoxResults.setStyle("-fx-border-color: black;");

78 **The -fx-border-radius Property**

- Specifies the radius size (amount of curvature of the corners) for a JavaFX node
- Format:
  -fx-border-radius: *pixels*;
  - *pixels* is an integer that specifies the amount of curvature
- Example:
  vBoxResults.setStyle("-fx-border-radius: 15;");

81 **The Slider Class (Page 1)**

- A Slider is a control used to indicate a continuous range of values between a given minimum and maximum
- The slider is rendered as a vertical or horizontal bar with a knob that the user can drag with the mouse to indicate the desired value

82 **The Slider Class (Page 2)**

- Format to instantiate:
  Slider *sliderObject* = new Slider( [*min*, *max*, *value*] );
  - *min* is lowest value for the slider
  - *max* is highest value for the slider
  - *value* is the *initial value* for the slider (and the location on the knob) when it first is instantiated
- Example:
  Slider sliderRate = new Slider(10.00, 25.00, 15.00);

83 **The setShowTickMarks Method**

- The method setShowTickMarks sets a Boolean value (true/false) indicating whether the *tick marks* for the Slider control are visible
- Format:
  *sliderObject*.setShowTickMarks( true|false );
- Example:
  sliderRate.setShowTickMarks(true);

84 **The setMajorTickUnit Method**

- The method setMajorTickCount for a slider sets how frequently "major" (larger) tick lines are shown
  - Also sets the locations where *labels* are shown (if labels are turned on)
- Format:
  *sliderObject*.setMajorTickUnit(*value*);
  - *value* is major tick line locations
- Example:
  sliderRate.setMajorTickUnit(3.0);

85 **The setMinorTickCount Method**

- The method setMinorTickCount for a slider sets the number of minor ticks to place

between any two major ticks

- Format:
  *sliderObject*.setMinorTickCount(*value*);
  - *value* is the number of minor ticks between two major ticks
- Example:
  sliderRate.setMinorTickCount(2);

86 **The setShowTickLabels Method**

- The method setShowTickLabels sets a Boolean value (true/false) indicating whether *labels* for the Slider are visible at the "major" tick units (see next slide)
- Format:
  *sliderObject*.setShowTickLabels( true|false );
- Example:
  sliderRate.setShowTickLabels(true);

87 **The getValue Method**

- The method getValue returns the value that represents the current location of the knob on the Slider
- Format:
  *sliderObject*.getValue();
- Example:
  labelRate.setText( sliderRate.getValue() );

88 **The setOnMouseDragged Method (Page 1)**

- The method setOnMouseDragged creates a MouseEvent handler for the "onMouseDragged" event of a JavaFX node
  - ActionEvent is only for Buttons, RadioButtons, CheckBoxes, etc.)
- For a Slider occurs every time user drags the knob
  - Method is inherited indirectly from class Node
- Can be used in a lambda expression

89 **The setOnMouseDragged Method (Page 2)**

- Format:
  *nodeObject*.setOnMouseDragged( e -> *methodName*(e) );
- Example:
  sliderRate.setOnMouseDragged( e -> updateRateSlider(e) );

90 **The setOnMouseClicked Method (Page 1)**

- The method setOnMouseClicked creates a MouseEvent handler for the "onMouseClicked" event of a JavaFX node
- For a Slider occurs every time user clicks on either side of the knob
  - Method is inherited indirectly from class Node
- Can be used in a lambda expression

91 **The setOnMouseClicked Method (Page 2)**

- Format:
  *nodeObject*.setOnMouseClicked( e -> *methodName*(e) );
- Example:
  sliderRate.setOnMouseClicked( e -> updateRateSlider(e) );

92 **The MouseEvent Class (Page 1)**

- Class MouseEvent stores event handler parameter information related to mouse events such as:
  - onMouseClicked—mouse button is pressed and released
  - onMousePressed—mouse button is pressed
  - onMouseReleased—mouse button is released
  - onMouseMoved—mouse moves within a node but no button pushed
  - onMouseEntered—mouse cursor "enters" and hovers over a node
  - onMouseExited—mouse "exits" a node
  - onMouseDragged—mouse is dragged (its button is pressed and held)

93 **The MouseEvent Class (Page 2)**

- Format as a parameter in an event handler method:
  *accessModifier* void *eventHandlerMethod*(MouseEvent event)
  { …
- Example:
  private void updateHoursSlider(MouseEvent event)
  { …

94 **The MouseEvent Class (Page 3)**

- Example as a parameter in a separate class event handler:

```
private class PayeeInputEventHandler implements EventHandler<MouseEvent>
{
   @Override
   public void handle(MouseEvent event)
   { …
   }
}
```

95 **The MouseEvent Class (Page 4)**

- Example as a parameter in an anonymous inner class event handler:

```
sliderRate.setOnMouseClicked(new EventHandler<MouseEvent>()
{
   @Override
   public void handle(MouseEvent event)
   { …
   }
}
)
```

96 **The DecimalFormat Class (Page 1)**

- Class used to create String objects used to *format numbers* for output
- Stored in the java.text package
  import java.text.DecimalFormat;
- Format:
  DecimalFormat *objectName* = new DecimalFormat("*formatString*");
  - *formatString* is String argument that specifies *how* numbers will be formatted

97 **The DecimalFormat Class (Page 2)**

- Example 1:
  DecimalFormat commaFormat = new DecimalFormat("#,##0");
- The String argument "#,##0" specifies that the number will display:
  - With *commas* at the thousands, millions, etc.
    - Only if number is *1000 or greater*; otherwise printing of leading zeros and commas are from the 10's position to the left are suppressed
  - Rounded to the nearest *integer*

98 **The DecimalFormat Class (Page 3)**

- Example 2:
  DecimalFormat twoDecimals = new DecimalFormat("0.00");
- The String argument "0.00" specifies that the number will display:
  - *At least one* digit to the left of the decimal point
  - *Exactly two digits* (rounded) to the right of the decimal point

99 **The DecimalFormat Class (Page 4)**

- The functionality of Examples 1 and 2 can be combined to add *commas* to the *two decimals* rounded:
  DecimalFormat grossPayFormat = new DecimalFormat("#,##0.00");
- A *floating dollar sign* could be inserted prior to the rest of the format string:
  DecimalFormat grossPayFormat = new DecimalFormat("$#,##0.00");

100 **The format Method**

- Returns a String that formats a numeric value according to the DecimalFormat object's *format string*
- Takes either a float or double as its *single* argument (may be a value, variable or expression)
- Format:
  *decimalFormatObject*.format(*float/double*);
- Example:
  JOptionPane.showMessageDialog( null, grossPayFormat.format(grossPay) );

102 **The ObservableList Class (Page 1)**

- The ObservableList is a collection class that is intended to be used with JavaFX collection objects like ListView

- Stored in the javafx.collections package
  import javafx.collections.ObservableList;

103 **The ObservableList Class (Page 2)**

- It is a generic class in that it always has a class *subtype*, e.g. ObservableList<String> or ObservableList<Book>
- Format to declare:
  ObservableList<*Type*> *object*;
  - ObservableList has *no constructors*
- Examples:
  ObservableList<String> titles;
  ObservableList<Book> books;

104 **The ObservableList Class (Page 3)**

- Since the subtype always must be a class, use a wrapper class to declare primitives
- Format:
  ObservableList<*WrapperType*> *object*;
- Example:
  ObservableList<Integer> pages;

105 **The observableArrayList Method (Page 1)**

- The static method observableArrayList is a "helper" method that populates an ObservableList object at the same time it is instantiated
- A member of the FXCollections class which is stored in the javafx.collections package
  import javafx.collections.FXCollections;

106 **The observableArrayList Method (Page 2)**

- Format:
  ObservableList<*Type*> *object* = FXCollections.observableArrayList(*items*);
  - *items* is a "list of elements" to be inserted into the ObservableList object that match its *Type*

107 **The observableArrayList Method (Page 3)**

- Example 1:
  ObservableList<Book> books = FXCollections.observableArrayList(book1, book2, book3, book4, book5, book6, book7, book8, book9, book10);

108 **The observableArrayList Method (Page 4)**

- Example 2:
  ObservableList<String> titles = FXCollections.observableArrayList( book1.getTitle(), book2.getTitle(), book3.getTitle(), book4.getTitle(), book5.getTitle(), book6.getTitle(), book7.getTitle(), book8.getTitle(), book9.getTitle(), book10.getTitle() );

109 **The add and addAll Methods (Page 1)**

- The add method for an ObservableList object inserts one additional element into the

collection after it is instantiated
- The addAll method for an ObservableList object inserts multiple additional elements into the collection after it is instantiated

110 **The add and addAll Methods (Page 2)**

- Formats:
  *observableListObject*.add(*element*);
  *observableListObject*.addAll(*elementList*);
- Examples:
  titles.add( book11.getTitle() );
  books.addAll(book11, book12, book13);

112 **The ListView Class (Page 1)**

- The ListView class creates a "scrollable" list of items from which the user selects one or more
  - Scroll bar activates if all items do not fit
- Stored in the javafx.scene.control package
  import javafx.scene.control.ListView;

113 **The ListView Class (Page 2)**

- Format to instantiate:
  ListView *object* = new ListView( [*observableListObject*] );
  - *observableListObject* inserted into the ListView are the items in the list
  - Items may be added afterwards by calling add or addAll methods
- Example:
  ListView listView = new ListView(titles);

114 **The add and addAll Methods (Page 1)**

- The add method for a ListView object inserts one additional element into the list after it is instantiated
- The addAll method for a ListView object inserts multiple additional elements into the list after it is instantiated
- Both methods are members of the items property of ListView which returns the current list of items in the list

115 **The add and addAll Methods (Page 2)**

- Formats:
  *listViewObject*.getItems.add(*item*);
  *listViewObject*.getItems.addAll(*itemList*);
- Examples:
  listView.getItems.add( book11.getTitle() );
  listView.getItems.addAll( book12.getTitle(), book13.getTitle(), book14.getTitle() );

116 **The getSelectedIndex Method (Page 1)**

- The getSelectedIndex method of the selectionModel property returns an int which is

the index of the currently selected item in a ListView object
  - Or any other JavaFX collection object
- Another member of sectionModel property is getSelectedItem which returns the "value" of the currently selected item

117 **The getSelectedIndex Method (Page 2)**

- Format:
  *listViewObject*.getSelectionModel().getSelectedIndex()
- Example:
  int index = listView.getSelectionModel().getSelectedIndex();

118 **The get Method**

- The get method of an ObservableList returns the element from the object that matches the index argument as its "generic type"
- Format:
  *observableListObject*.get(*index*)
  - *index* is the location in the collection
- Example:
  labelResults.setText( books.get(index).toString() );

121 **The ComboBox Class (Page 1)**

- The ComboBox class creates a "drop-down" list of items from which the user selects one or more
  - Scroll bar activates if all items do not display when ComboBox drops down
- Stored in the javafx.scene.control package
  import javafx.scene.control.ComboBox;

122 **The ComboBox Class (Page 2)**

- Format to instantiate:
  ComboBox *object* = new ComboBox( [*observableListObject*] );
  - *observableListObject* inserted into the ComboBox are the items in the list
  - Items may be added afterwards by calling add or addAll methods
- Example:
  ComboBox comboBoxEmail = new ComboBox(emailAddresses);

123 **The add and addAll Methods (Revisited)**

- The add and addAll methods of ListView also apply to ComboBox
- Formats:
  *comboBoxObject*.getItems.add(*item*);
  *comboBoxObject*.getItems.addAll(*itemList*);
- Examples:
  comboBoxPriorities.getItems.add("Highest");
  comboBoxPriorities.getItems().addAll("Highest", "High", "Normal", "Low", "Lowest");

124 **The setVisibleRowCount Method (Page 1)**

- The setVisibleRowCount method sets the visibleRowCount property which is the maximum number of rows displayed when the ComboBox drop-down list is showing
  - Default value is 10
- A scroll bar automatically will be activated if there are more items than are displayed

125 **The setVisibleRowCount Method (Page 2)**

- Format:
  *comboBoxObject*.setVisibleRowCount(*count*);
  - The *count* is the number of rows visible when ComboBox drops down
- Example:
  comboBoxEmail.setVisibleRowCount(5);

126 **The setEditable Method (Revisited) (Page 1)**

- The function of a ComboBox often includes a text input area for users to type their own values into the box instead of selecting one of the items from the list
  - Both selecting an item from the drop-down list and typing a new value updates its value property
- The setEditable method sets the editable property which, when set to true, allows for user typed input
  - Default value is false

127 **The setEditable Method (Revisited) (Page 2)**

- Format:
  *comboBoxObject*.setEditable(true | false);
  - The *count* is the number of rows visible when ComboBox drops down
- Example:
  comboBoxEmail.setEditable(true);

128 **The setOnAction Method (Revisited) (Page 1)**

- Event handling for the ComboBox is managed by an ActionEvent which "fires" when selecting a new value from the list
- The setOnAction method (the same as used for Button "click" events) is used to name an event handler that executes when a new item is selected

129 **The setOnAction Method (Revisited) (Page 2)**

- Format:
  ComboBox *comboBoxObject* = new ComboBox( ... );
  *comboBoxObject*.setOnAction( e -> *methodCall*( *parameters* ) );
- Example
  ComboBox comboBoxEmail = new ComboBox(emailAddresses);
  comboBoxEmail.setOnAction( e -> newValueSelected(e) );

130 **The getValue Method**

- The getValue method of a ComboBox returns the "value" of the currently selected item as the "generic type" of the ComboBox

- Format:
  *comboBoxObject*.getValue()
- Example:
  String priority = comboBoxPriorities.getValue();

131 **The getSelectedItem Method**

- Additionally, the getSelectedItem method of the selectionModel property also returns the "value" of the currently selected item as the "generic type" of the ComboBox
- Format:
  *comboBoxObject*.getSelectionModel().getSelectedValue()
- Example:
  String priority = comboBoxPriorities.getSelectionModel.getSelectedItem();

132 **The getSelectedIndex Method (Revisited)**

- Method getSelectedIndex of ListView that returns the int index of the currently selected element also applies to ComboBox
- Format:
  *comboBoxObject*.getSelectionModel().getSelectedIndex()
- Example:
  int index = comboBox.getSelectionModel().getSelectedIndex();