Application Program Development

Segment : Working with Scene Builder

Mahboob Ali

Outcomes

- Understanding Scene builder more
- Using different Nodes, Controls and Widgets

Outcomes

- Understanding Scene builder more
- Using different Nodes, Controls and Widgets

Example Preparation

- We are going to experience another example by using
  - Additional Layouts
  - Additional Controls
  - Mouse events with radio button events
  - Binding events with properties
  - Shapes like rectangles and circles

Scene Graph and Node Size

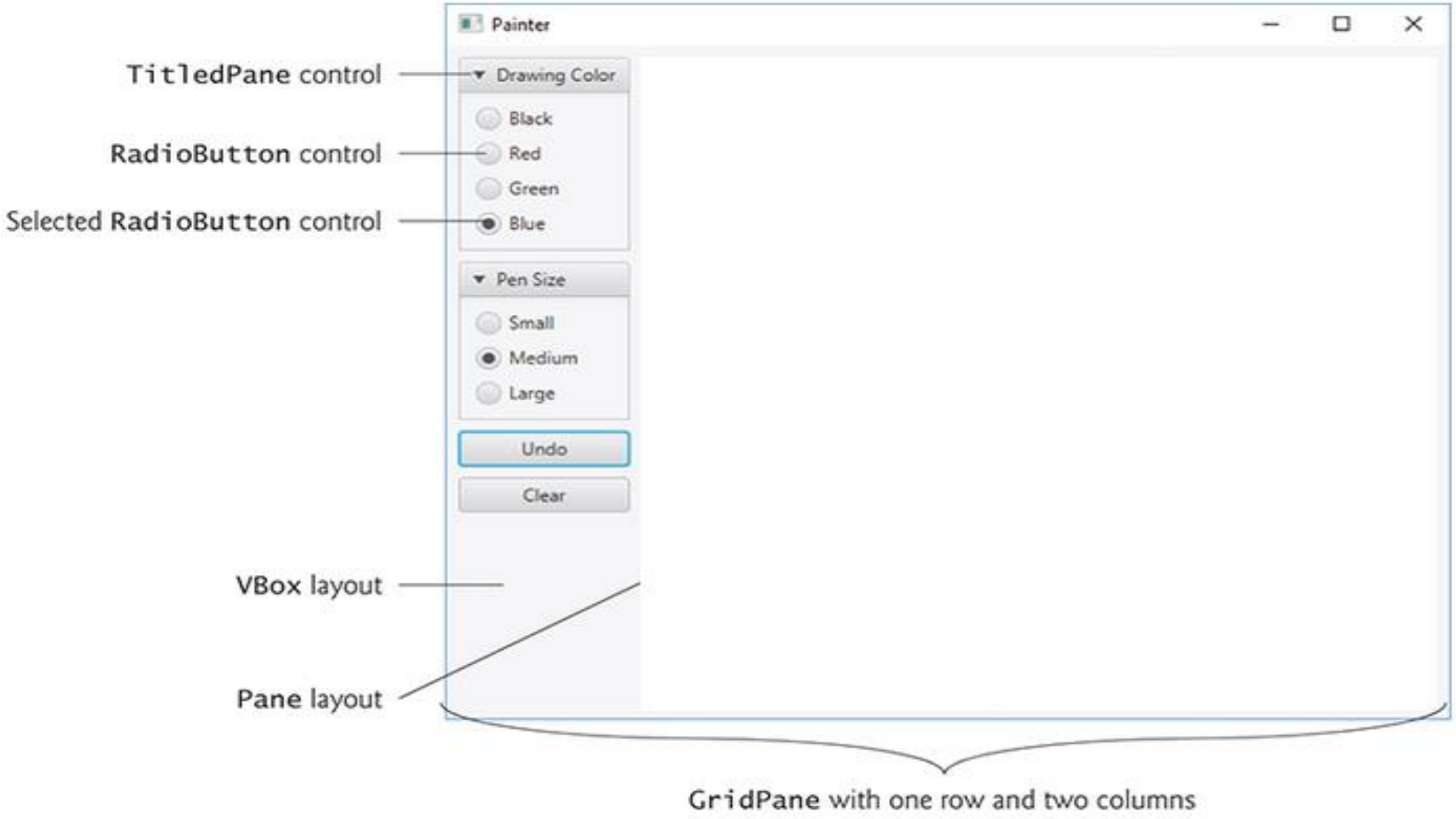
- Nodes shouldn't be defined with explicit size.
- Disadvantage of this when the window size will change the position of the nodes will be changed as well.
- Instead of using width and height properties, we can specify node's range with acceptable size.
  - The preferred size properties specify a node's preferred width and height that should be used by the layout in most cases.

Scene Graph and Node Position

- A node's position should be defined *relative* to its parent node and the other nodes in its parent.

- 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.

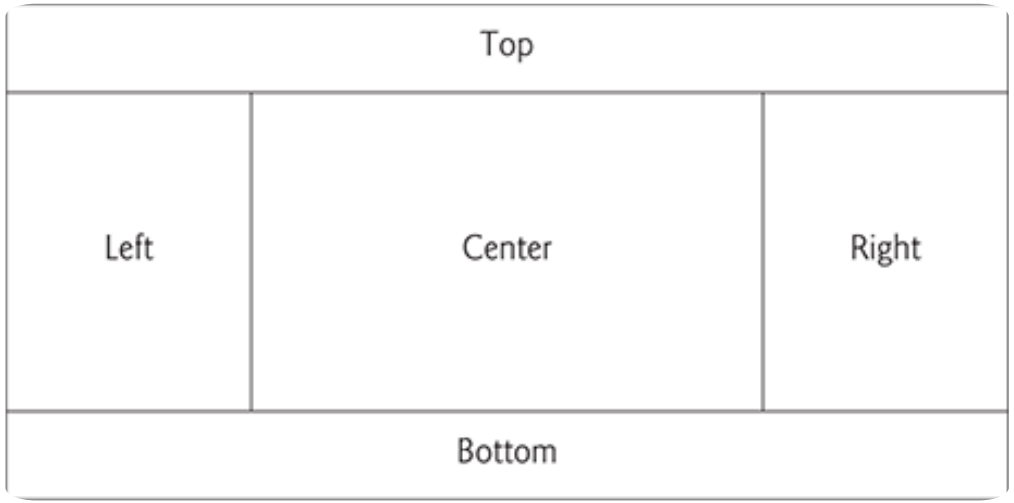
Painting Desktop Application



Painting Desktop Application

- First create the JavaFX project in the eclipse by following the last week lecture steps.
- Project Name: PainterApplication
- FXML File Name: Painter.fxml
- Once the project is created open the Painter.fxml in the scene builder.
- Drag a BorderPane from the Scene Builder Library window's Containers section onto the content panel.

BorderPane



- All the areas in the BorderPane are optional
  - *If the top or bottom area is empty, the left, center and right areas expand vertically to fill that area.*
  - *If the left or right area is empty, the center expands horizontally to fill that area.*

Painting Desktop Application

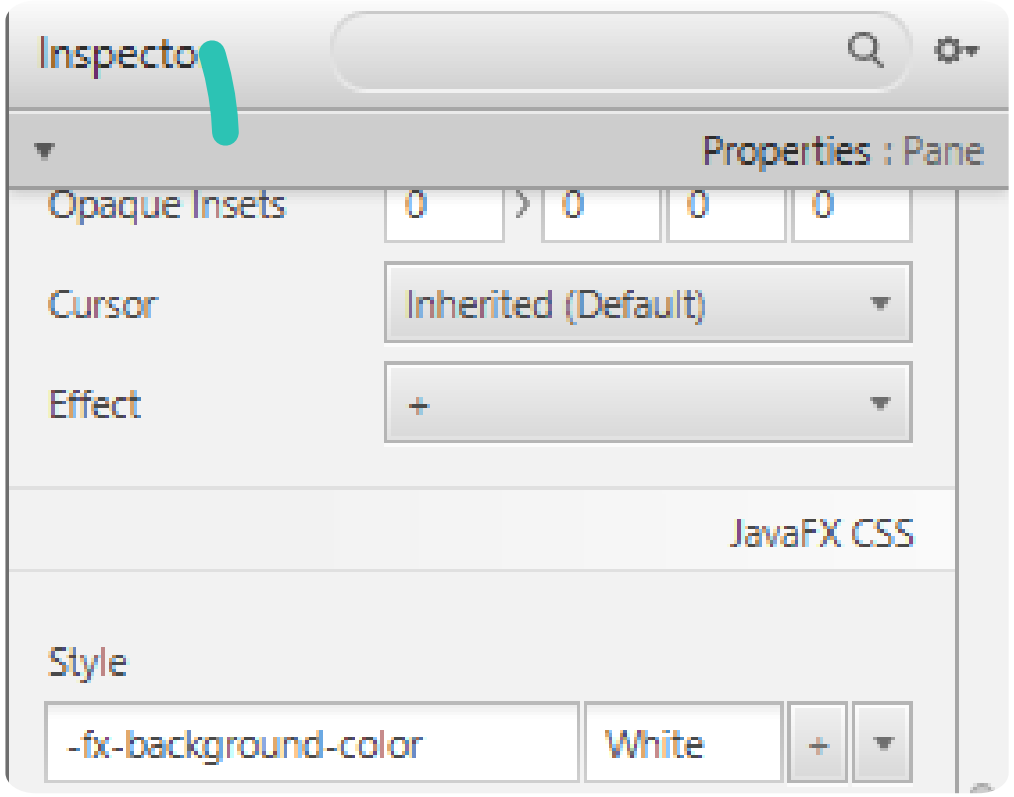
- Set the PrefWidth property to 640.
- Set the PrefHeight property to 480.
- Drag a VBox into BorderPane's left area
- Drag a Pane into BorderPane's center area.
- Set the Pane's fx:id to drawingAreaPane

VBox properties

- Set the spacing property to 8, this will set the default vertical spacing between the controls to 8 for all. (consistency in the design)
- Set the Margin property to 8, this will add horizontal spacing.
- Set the PrefWidth and PrefHeight to 'USE_COMPUTED_SIZE'.
- Set the MaxHeight to 'MAX_VALUE', this will adjust the height of VBox to be wide as its needed.

Pans's properties

- Reset the Pane's Pref Width and Pref Height to their default USE_COMPUTED_SIZE values.
- Set its Max Width and Max Height to MAX_VALUE so that it occupies the full width and height of the BorderPane's center area.
- In the JavaFX CSS category of the Inspector window's Properties section, click the field below Style (which is initially empty) and select -fx-background-color to indicate that you'd like to specify the Pane's background color. In the field to the right, specify white.



TitledPane Layout Container

- A TitledPane layout container displays a title at its top and is a collapsible panel containing a layout node.

- This in turn will contains other nodes.

- We'll use TitledPanes to organize the app's RadioButtons and to help the user understand the purpose of each RadioButton group.

TitledPane Properties

- Drag two TitledPane (empty) objects onto VBox.
  - Set the first's TitledPane Text property to Drawing.
  - Set the Second TitledPane Text property to Pen Size.

- We will use VBox in each of the TitledPane to arrange our nodes in them.
  - Drag a VBox and drop it in each TitledPane.
    - Set Spacing property for each VBox to 8.
    - Set PrefWidth and PrefHeight for each VBox to USE_COMPUTED_SIZE**.

**USE_COMPUTED_SIZE: will have the nodes sized based on the available spacing in the layout.

RadioButton and ToggleGroup

- 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.
- We'll use JavaFX Scene Builder's capability for specifying each RadioButton's ToggleGroup in FXML;
  - But one can also create a ToggleGroup in Java, then use a RadioButton's setToggleGroup method to specify its ToggleGroup.

RadioButton and Properties

- Drag four RadioButtons onto the VBox for the Drawing Color TitledPane.

- Text properties and fx:ids,

Text	fx:id's
Black	blackRadioButton
Red	redRadioButton
Green	greenRadioButton
Blue	blueRadioButton

RadioButton and Properties

- Drag three RadioButtons onto the VBox for the Pen Size TitledPane
- Text properties and fx:ids,

Text	fx:id's
Small	smallRadioButton
Medium	mediumRadioButton
Large	largeRadioButton

- Select the blackRadioButton and ensure that its Selected property is checked, then do the same for the mediumRadioButton.

ToggleGroup's properties for RadioButton's

- Select all four RadioButtons in the first TitledPane's VBox, then set the Toggle Group property to colorToggleGroup.

- Once the FXML file is loaded, a ToggleGroup object by that name will be created and these four RadioButtons will be associated with it to ensure that only one is selected at a time.

- Repeat this step for the three RadioButtons in the second TitledPane's VBox, but set the Toggle Group property to sizeToggleGroup.

TiltedPane properties

- For each TitledPane, set its Pref Width and Pref Height to 'USE_COMPUTED_SIZE' so the TitledPanes will be sized based on their contents.

- This will eliminate all the extra space and make the design neater and cleaner.

Adding Button's

- Add two Buttons below the TitledPanes.
- Text properties and fx:ids,

Text	fx:id's
Undo	undoButton
Clear	clearButton

- Set each Button's Max Width property to MAX_VALUE so that they fill the VBox's width.

VBox properties

- We'd like the VBox to be only as wide as it needs to be to display the controls in that column.
- To specify this, select the VBox in the Document window's Hierarchy section. Set the properties as follows
  - Set the column's Min Width and Pref Width to USE_COMPUTED_SIZE
  - Set the Max Width to USE_PREF_SIZE**
  - Reset the Max Height to its default USE_COMPUTED_SIZE value.

**USE_PREF_SIZE : Indicates that the maximum width should be the preferred width

Controller Class

- JavaFX FXML app, the app's controller class typically defines instance variables for interacting with controls programmatically, as well as event-handling methods.
- To ensure that an object of the controller class is created when the app loads the FXML file at runtime, we must specify the controller class's name in the FXML file:
- Expand Scene Builder's Controller window (located below the Hierarchy window).
- In the Controller Class field, type PainterController.

Event-Handler Methods Names

- For the drawingAreaPane, specify drawingAreaMouseDragged as the On Mouse Dragged event handler.
  - This method will draw a circle in the specified color and size for each mousedragged event.
- For the four Drawing Color RadioButtons (Select all the RadioButton's), specify colorRadioButtonSelected as each RadioButton's On Action event handler.
  - This method will set the current drawing color, based on the user's selection.
- For the three Pen Size RadioButtons (Select all the RadioButton's), specify sizeRadioButtonSelected as each RadioButton's On Action event handler.
  - This method will set the current pen size, based on the user's selection.
- For the Undo Button, specify undoButtonPressed as the On Action event handler.
  - This method will remove the last circle the user drew on the screen.
- For the Clear Button, specify clearButtonPressed as the On Action event handler.
  - This method will clear the entire drawing.

Generating FXML Controller Class Methods

- Scene Builder generates the initial controller-class skeleton as well
- Select

  View > Show Sample Controller Skeleton.

- You can copy this code into a PainterController.java file and store the file in the same folder as Painter.fxml

Main.java

```
package PainterApplication;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {
	@Override
	public void start(Stage primaryStage) {
		try {
			Parent root =
FXMLLoader.load(getClass().getResource("Painter.fxml"));
			Scene scene = new Scene(root);
			primaryStage.setTitle("Painter");
			primaryStage.setScene(scene);
			primaryStage.show();
		} catch(Exception e) {
				e.printStackTrace();
		}
	}
	public static void main(String[] args) { launch(args); }
}
```

PainterController.java

```
public class PainterController {

    //enum to represent penSizes
    private enum PenSize{
    SMALL(2),
    MEDIUM(4),
    LARGE(6);

    private final int radius;

    //constructor for the enum Inner class
    PenSize(int radius){this.radius = radius;}
    public int getRadius() {
        return this.radius;
    }
};
```

PainterController.java

Continue…

```
// instance variables that refer to GUI components
@FXML private RadioButton blackRadioButton;
@FXML private RadioButton blueRadioButton;
@FXML private Button clearButton;
@FXML private ToggleGroup colorToggleGroup;
@FXML private Pane drawingAreaPane;
@FXML private RadioButton greenRadioButton;
@FXML private RadioButton largeRadioButton;
@FXML private RadioButton mediumRadioButton;
@FXML private RadioButton redRadioButton;
@FXML private ToggleGroup sizeToggleGroup;
@FXML private RadioButton smallRadioButton;
@FXML private Button undoButton;
```

PainterController.java

Continue…

```
// instance variables for managing Painter state
private PenSize radius = PenSize.MEDIUM; // radius of circle
private Paint brushColor = Color.BLACK; // drawing color

// set user data for the RadioButtons
public void initialize() {
   // user data on a control can be any Object
   blackRadioButton.setUserData(Color.BLACK);
   redRadioButton.setUserData(Color.RED);
   greenRadioButton.setUserData(Color.GREEN);
   blueRadioButton.setUserData(Color.BLUE);
   smallRadioButton.setUserData(PenSize.SMALL);
   mediumRadioButton.setUserData(PenSize.MEDIUM);
   largeRadioButton.setUserData(PenSize.LARGE);
}
```

PainterController.java

Continue…

```
// handles drawingArea's onMouseDragged MouseEvent
@FXML
void drawingAreaMouseDragged(MouseEvent event) {
        Circle newCircle = new Circle(event.getX(),event.getY(),
                                      radius.getRadius(), brushColor);
        drawingAreaPane.getChildren().add(newCircle);
}
// handles color RadioButton's ActionEvents
@FXML
void colorRadioButtonSelected(ActionEvent event) {
// user data for each color RadioButton is the corresponding Color
brushColor = (Color) colorToggleGroup.getSelectedToggle().getUserData();
}
```

PainterController.java

Continue…

```
 // handles size RadioButton's ActionEvents
@FXML
private void sizeRadioButtonSelected(ActionEvent e) {
    // user data for each size RadioButton is the corresponding PenSize
       radius = (PenSize) sizeToggleGroup.getSelectedToggle().getUserData();
}
// handles Undo Button's ActionEvents
@FXML
private void undoButtonPressed(ActionEvent event) {
      int count = drawingAreaPane.getChildren().size();

     // if there are any shapes remove the last one added
     if (count > 0) {
          drawingAreaPane.getChildren().remove(count - 1);
        }
    }
// handles Clear Button's ActionEvents
@FXML
private void clearButtonPressed(ActionEvent event) {
    drawingAreaPane.getChildren().clear(); // clear the canvas
}
```