OBJECT-ORIENTED PROGRAMMING

10. GUI PROGRAMMING WITH JAVAFX

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

1

Outline

2

Review: JavaFX vs. Swing

- **JavaFX** is a software platform for creating and delivering desktop applications, as well as **rich internet applications** (RIAs) that can run across a wide variety of devices.
  - **Rich Internet applications** (**RIA**) are **Web**-based **applications** that have some characteristics of graphical desktop **applications**
- **JavaFX** is intended to replace **Swing** as the standard ***GUI library*** *for* ***Java SE***, but both will be included for the foreseeable future.

3

Why JavaFX?

- Additional consistent in its style than Swing
- Design GUI like Web apps with XML and CSS (FXML) than doing in Java code
  - Save building time
- Integrate 3D graphics, charts, and audio, video, and embedded Website inside GUI
  - Easy to develop Game/Media applications
- Light-weight and hardware accelerated
- Contains WebView supported the popular WebKit browser
  - Introduce Website within JavaFX

4

Why JavaFX? (2)

- Support stylish animations
  - Resembling fades, Rotations, Motion ways
  - Custom animations with KeyFrame and Timeline
- Support for modern touch devices
  - Resembling scrolling, swipping, rotating and zooming…
- Many eye-catching controls
  - E.g. Collapsible Titled Pane
- Events are higher thought-out and additional consistent

5

Outline

1. JavaFX Overview
2. MVC Architectural Pattern
3. JavaFX Window Structure
4. Binding Properties
5. Data-Driven UIs

6

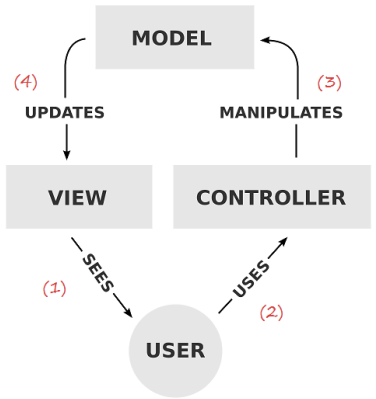
MVC Architectural Pattern

- Architectural Pattern
  - General and reusable solution to a commonly occurring problem in software architecture within a given context
- Model-View-Controller (MVC)
  - Dividing the related program logic into three interconnected elements
  - Separating **internal representations** of information from **the ways** information is **presented to and accepted from the user**

7

MVC pattern

1. After seeing it on VIEW
2. Users use CONTROLLER
3. Manipulate data (Update, modify, delete, ..), the data on MODEL has been changed.
4. Displaying the data of MODEL on VIEW.



8

Model-View-Controller

- Model
  - Manageing the data of the application, independent of the user interface
- View
  - Rendering presentation of the data (from model) in a particular format to user
- Controller
  - Accepting and responding to the user input
    - logic to process user inputs
  - Performs interactions on the data model objects
  - Updating views and/or showing data on views

9

Controller
- responds to events
- accesses model / view
- propagates data

Model and view

Model
- long term storage
- data members
- domain logic

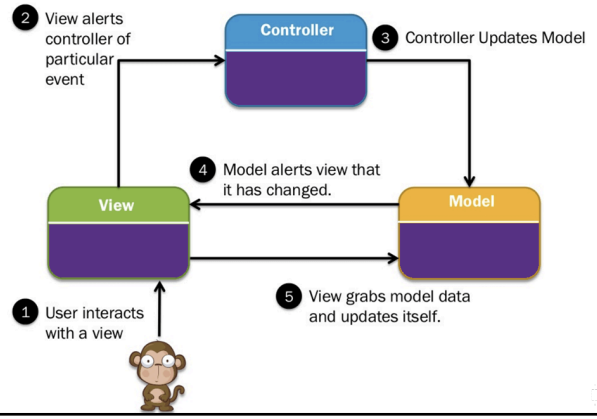
View
- tables to display
- forms to change
- controls to listen for events

Model-View-Controller

Should probably be Model-Controller-View

10

MVC in JavaFX



11

JavaFX View: FXML

Records the widgets in the GUI, their visible attributes and their relationship to each other

An XML vocabulary for defining and arranging JavaFX GUI controls without writing any Java code

MyScene.fxml

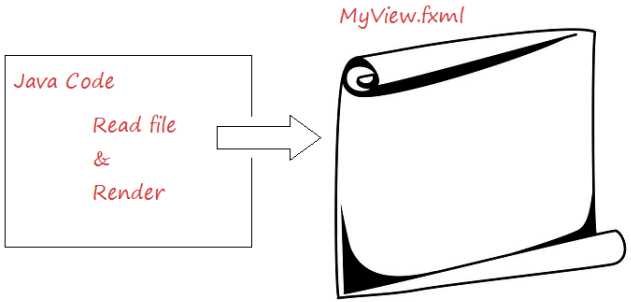
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="238.0" prefWidth="269.0"
    xmlns="http://javafx.com/javafx/8"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="org.o7planning.javafx.MyController">

   <children>

      <Button fx:id="myButton" layoutX="51.0" layoutY="44.0"
           mnemonicParsing="false"
           onAction="#showDateTime" text="Show Date Time" />

      <TextField fx:id="myTextField" layoutX="28.0"
           layoutY="107.0" prefHeight="25.0" prefWidth="201.0" />

   </children>

</AnchorPane>
```

12

13

Building View: JavaFX Scene Builder

- A GUI builder called SceneBuilder allows drag-and-drop manipulation of widgets



13

14

JavaFX Controller: a Java class

- Can interact with UI controls of FXML
  - Controller class name must be the same as *"fx:controller"* attribute of the main panel in FXML
  - Must declare UI control with name the same as the attribute *"fx:id"* of that UI control in FXML
- An **event handler** for the view
  - When the user interacts with a control, the control generates an event
  - Must provide event handling methods corresponding to declarations in the FXML

  (e.g. onAction="#showDateTime")

14

Controller: Java code

15

```
public class MyController implements Initializable {
    @FXML
    private Button myButton;
    @FXML
    private TextField myTextField;

    // When user click on myButton, this method will be called
    public void showDateTime(ActionEvent event) {
        System.out.println("Button Clicked!");
        Date now= new Date();
        DateFormat df = new SimpleDateFormat(
                            "dd-MM-yyyy HH:mm:ss.SSS");
        // Model Data
        String dateTimeString = df.format(now);
        // Show in VIEW
        myTextField.setText(dateTimeString);
    }
}
```

Must be completed by the programmer to bring the GUI to life

15

16

Discussion

- What are the benefits to separate the UI (View) and the program logic (Controller)?

16

Additional elements

- A set of classes to describe the model, which is what the GUI allows the user to interact with
- A set of cascading style sheets (CSS files) to further secify “look-and-feel”

18

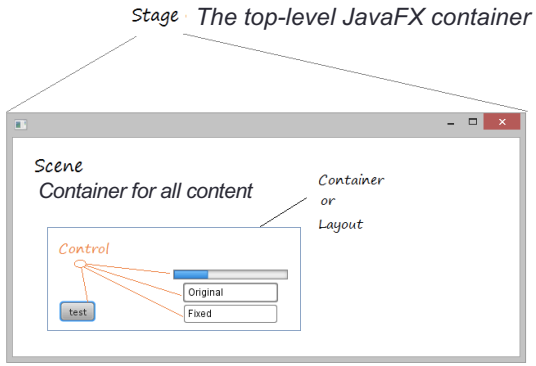
Outline

1. JavaFX Overview
2. MVC Architectural Pattern
3. JavaFX Window Structure
4. Binding Properties
5. Data-Driven UIs

19

Basic Structure of JavaFX

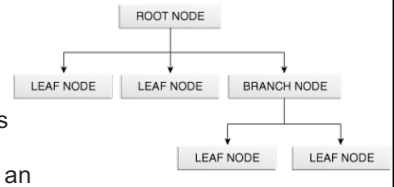
Swing?

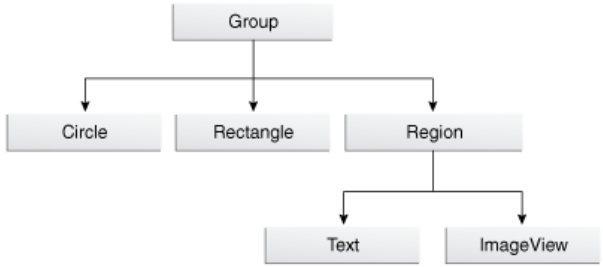


20

Scene graph

- A JavaFX user interface is based on a scene graph, which is a tree, much like an HTML document

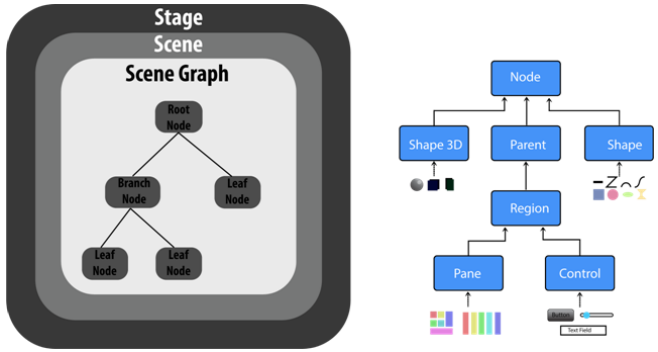




21

22

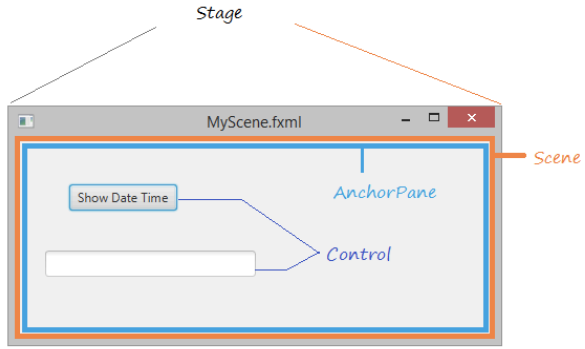
Basic Structure of JavaFX



22

23

E.g. Show Date Time UI



23

24

MyScene.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.text.*?>
4  <?import javafx.scene.control.*?>
5  <?import java.lang.*?>
6  <?import javafx.scene.layout.*?>
7  <?import javafx.scene.layout.AnchorPane?>
8
9  <AnchorPane prefHeight="238.0" prefWidth="269.0"
10     xmlns="http://javafx.com/javafx/8"
11     xmlns:fx="http://javafx.com/fxml/1"
12     fx:controller="org.o7planning.javafx.MyController">
13
14    <children>
15
16       <Button fx:id="myButton" layoutX="51.0" layoutY="44.0"
17            mnemonicParsing="false"
18            onAction="#showDateTime" text="Show Date Time" />
19
20       <TextField fx:id="myTextField" layoutX="28.0"
21            layoutY="107.0" prefHeight="25.0" prefWidth="201.0" />
22
23    </children>
24
25 </AnchorPane>
```

24

```
public class MyApplication  extends Application {


    @Override
    public void start(Stage primaryStage) {
       try {
          // Read file fxml and draw interface.
          Parent root = FXMLLoader.load(getClass()
                .getResource("/org/o7planning/javafx/MyScene.fxml"));

          primaryStage.setTitle("My Application");
          primaryStage.setScene(new Scene(root));
          primaryStage.show();

       } catch(Exception e) {
          e.printStackTrace();
       }
    }

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

}
```

25

FXMLLoader: load() method

- Uses the FXML file that represents the app's GUI to create the GUI's scene graph
- Returns a Parent (package javafx.scene) reference to the scene graph's root node
- Initializes the controller's instance variables,
- Creates and registers the event handlers for any events specified in the FXML

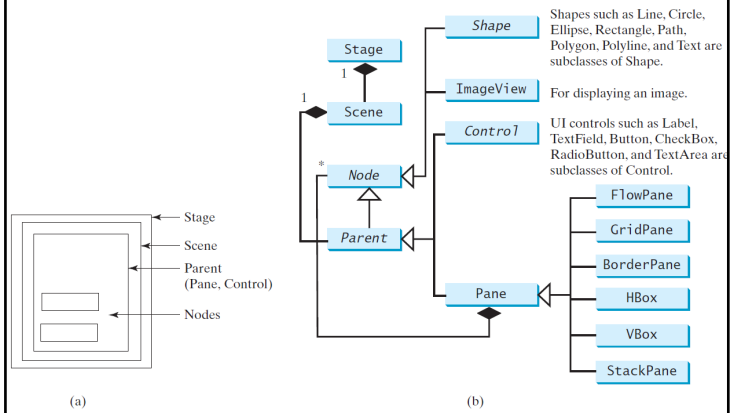
26

Exercise: Show current date time

- Please create a JavaFX application to show the current date time to a text field when the user click on a button with text "Show Date Time"

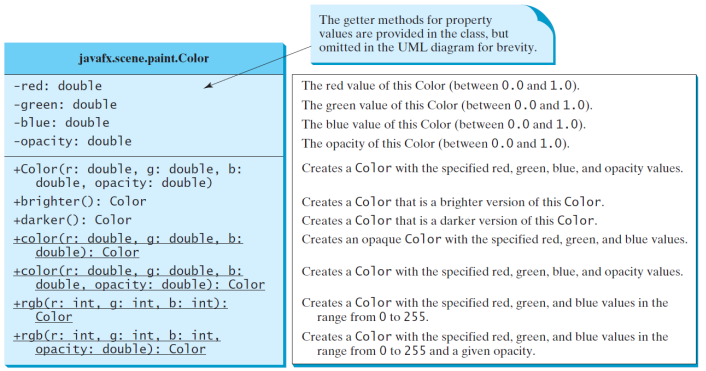
27

Panes, UI Controls, and Shapes



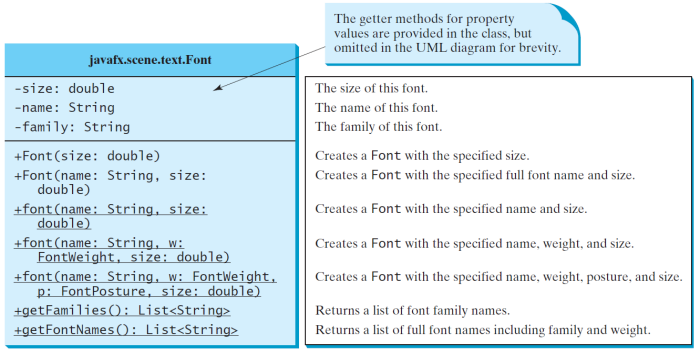
28

The Color Class



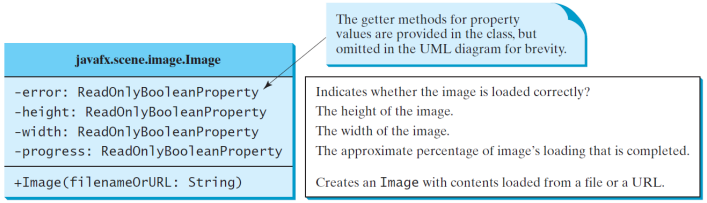
29

The Font Class



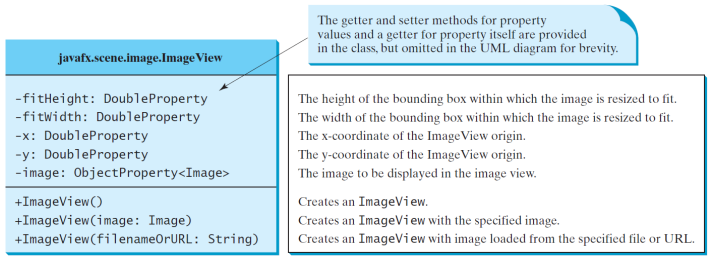
30

The Image Class



31

The ImageView Class



32

Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

33

FlowPane

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.FlowPane

```
-alignment: ObjectProperty<Pos>
-orientation:
   ObjectProperty<Orientation>
-hgap: DoubleProperty
-vgap: DoubleProperty
```

```
+FlowPane()
+FlowPane(hgap: double, vgap:
   double)
+FlowPane(orientation:
   ObjectProperty<Orientation>)
+FlowPane(orientation:
   ObjectProperty<Orientation>,
   hgap: double, vgap: double
```

The overall alignment of the content in this pane (default: `Pos.LEFT`).
The orientation in this pane (default: `Orientation.HORIZONTAL`).

The horizontal gap between the nodes (default: `0`).
The vertical gap between the nodes (default: `0`).

Creates a default `FlowPane`.
Creates a `FlowPane` with a specified horizontal and vertical gap.

Creates a `FlowPane` with a specified orientation.

Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

GridPane

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane

```
-alignment: ObjectProperty<Pos>
-gridLinesVisible:
   BooleanProperty
-hgap: DoubleProperty
-vgap: DoubleProperty
```

```
+GridPane()
+add(child: Node, columnIndex:
   int, rowIndex: int): void
+addColumn(columnIndex: int,
   children: Node...): void
+addRow(rowIndex: int,
   children: Node...): void
+getColumnIndex(child: Node):
   int
+setColumnIndex(child: Node,
   columnIndex: int): void
+getRowIndex(child:Node): int
+setRowIndex(child: Node,
   rowIndex: int): void
+setHalighnment(child: Node,
   value: HPos): void
+setValighnment(child: Node,
   value: VPos): void
```

The overall alignment of the content in this pane (default: `Pos.LEFT`).
Is the grid line visible? (default: `false`)

The horizontal gap between the nodes (default: 0).
The vertical gap between the nodes (default: 0).

Creates a `GridPane`.
Adds a node to the specified column and row.

Adds multiple nodes to the specified column.

Adds multiple nodes to the specified row.

Returns the column index for the specified node.

Sets a node to a new column. This method repositions the node.

Returns the row index for the specified node.
Sets a node to a new row. This method repositions the node.

Sets the horizontal alignment for the child in the cell.

Sets the vertical alignment for the child in the cell.

ShowGridPane

Run

BorderPane

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.BorderPane

```
-top: ObjectProperty<Node>
-right: ObjectProperty<Node>
-bottom: ObjectProperty<Node>
-left: ObjectProperty<Node>
-center: ObjectProperty<Node>
```

```
+BorderPane()
+setAlignment(child: Node, pos:
   Pos)
```

The node placed in the top region (default: `null`).
The node placed in the right region (default: `null`).
The node placed in the bottom region (default: `null`).
The node placed in the left region (default: `null`).
The node placed in the center region (default: `null`).

Creates a `BorderPane`.
Sets the alignment of the node in the `BorderPane`.

HBox

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.HBox

```
-alignment: ObjectProperty<Pos>
-fillHeight: BooleanProperty
-spacing: DoubleProperty
```

```
+HBox()
+HBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full height of the box (default: `true`).
The horizontal gap between two nodes (default: `0`).

Creates a default `HBox`.
Creates an `HBox` with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

VBox

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.VBox

```
-alignment: ObjectProperty<Pos>
-fillWidth: BooleanProperty
-spacing: DoubleProperty
```

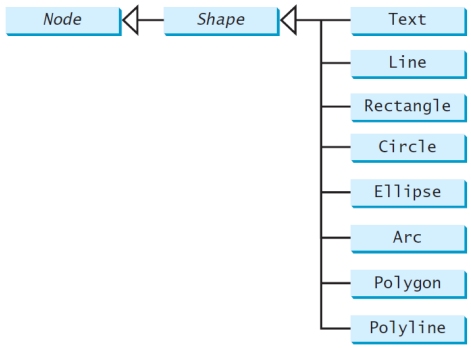
```
+VBox()
+VBox(spacing: double)
+setMargin(node: Node, value: Insets): void
```

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full width of the box (default: `true`).
The vertical gap between two nodes (default: 0).

Creates a default `VBox`.
Creates a `VBox` with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

Shapes

JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.



Text

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Text

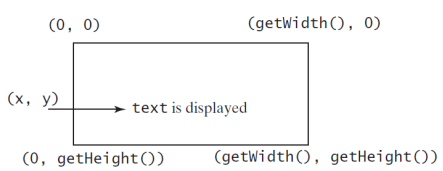
```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+Text()
+Text(text: String)
+Text(x: double, y: double, text: String)
```

Defines the text to be displayed.
Defines the x-coordinate of text (default 0).
Defines the y-coordinate of text (default 0).
Defines if each line has an underline below it (default `false`).
Defines if each line has a line through it (default `false`).
Defines the font for the text.

Creates an empty Text.
Creates a Text with the specified text.
Creates a Text with the specified x-, y-coordinates and text.

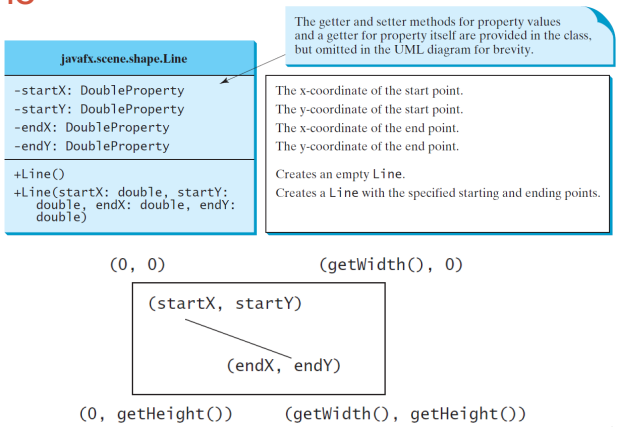
Text Example



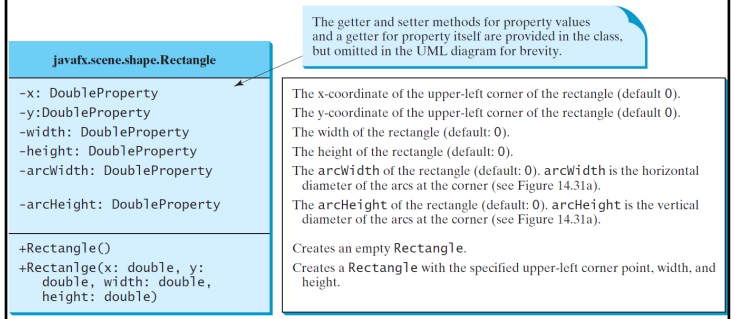
(a) Text(x, y, text)

(b) *Three* Text *objects are displayed*

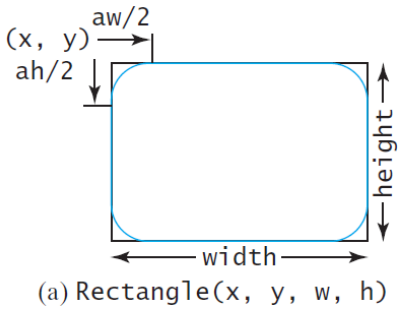
Line



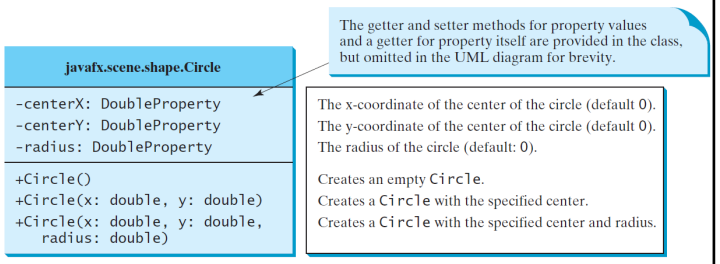
Rectangle



Rectangle Example



Circle



Ellipse

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

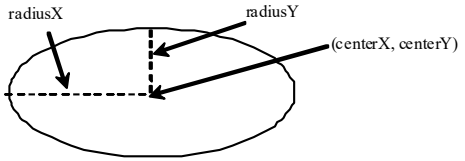
javafx.scene.shape.Ellipse

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
```

```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).

Creates an empty `Ellipse`.
Creates an `Ellipse` with the specified center.
Creates an `Ellipse` with the specified center and radiuses.



Arc

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Arc

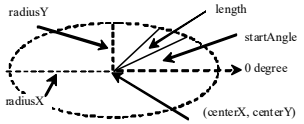
```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

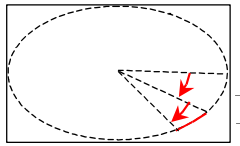
```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

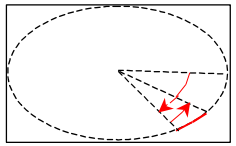
Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.

Arc Examples



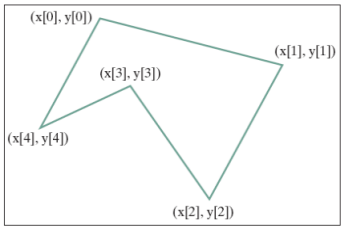


(a) Negative starting angle –30° and negative spanning angle –20°

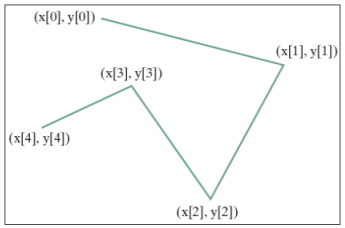


(b) Negative starting angle –50° and positive spanning angle 20°

Polygon and Polyline



(a) Polygon



(b) Polyline

javafx.scene.shape.Polygon

```
+Polygon()
+Polygon(double... points)
+getPoints():
   ObservableList<Double>
```

The `getter` and `setter` methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

Creates an empty polygon.
Creates a polygon with the given points.
Returns a list of double values as x- and y-coordinates of the points.

Outline

1. JavaFX Overview
2. MVC Architectural Pattern
3. JavaFX Window Structure
4. Binding Properties
5. Data-Driven UIs

Binding properties

- JavaFX introduces a new concept called *binding property* that enables a *target object* to be bound to a *source object*
- If the value in the source object changes, the target property is also changed automatically
- The target object is simply called a *binding object* or a *binding property*

Example: Binding Properties

Unidirectional Binding

```
lable1.textProperty().bind(text1.textProperty());
```

- text1 changes, lable1 changes correspondingly
- label1 changes, text1 does not change

Bidirectional Binding

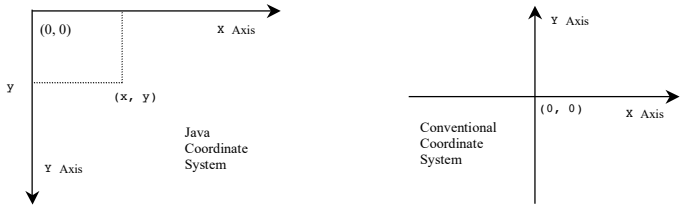
```
text1.textProperty().
      bindBidirectional(text2.textProperty());
```

- text1 changes, text2 changes correspondingly
- text2 changes, text1 changes correspondingly

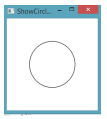
Exercise: How to bind properties for the below application?

Display a Shape

This example displays a circle in the center of the pane.

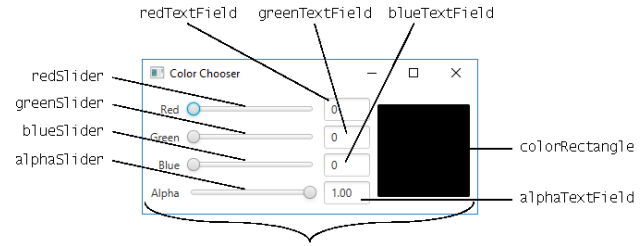



```
public class ShowCircleCentered extends Application {
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new Pane();
    Circle circle = new Circle();

circle.centerXProperty().bind(pane.widthProperty().divide(2));
circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle); // Add circle to the pane

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircleCentered");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```



Binding Properties

Example: ColorChooser



Building the GUI

- Step 1: Adding a GridPane
  - Drag a GridPane from the Library window's Containers section onto Scene Builder's content panel
- Step 2: Configuring the GridPane
  - This app's GridPane requires four rows and four columns
  - Use the techniques you've learned previously to add two columns and one row to the GridPane
  - Set the GridPane's Hgap and Padding properties to 8 to inset the GridPane from the stage's edges and to provide space between its columns

Building the GUI (cont.)

- Step 3: Adding the Controls
  - Add the Labels, Sliders, TextFields, a Circle and a Rectangle to the GridPane—Circle and Rectangle are located in the Scene Builder Library's Shapes section
  - When adding the Circle and Rectangle, place both into the rightmost column's first row
  - Be sure to add the Circle before the Rectangle so that it will be located behind the rectangle in the layout
  - Set the text of the Labels and TextFields as shown and set all the appropriate fx:id properties as you add each control.

Building the GUI (cont.)

- Step 4: Configuring the Sliders
  - For the red, green and blue Sliders, set the Max properties to 255 (the maximum amount of a given color in the RGBA color scheme)
  - For the alpha Slider, set its Max property to 1.0 (the maximum opacity in the RGBA color scheme).
- Step 5: Configuring the TextFields
  - Set all of the TextField's Pref Width properties to 50.

Building the GUI (cont.)

- Step 6: Configuring the Rectangle
  - Set the Rectangle's Width and Height properties to 100, then set its Row Span property to Remainder so that it spans all four rows.
- Step 7: Configuring the Circle
  - Set the Circle's Radius property to 40, then set its Row Span property to Remainder so that it spans all four rows.

Building the GUI (cont.)

- Step 8: Configuring the Rows
  - Set all four columns' Pref Height properties to USE_COMPUTED_SIZE so that the rows are only as tall as their content
- Step 9: Configuring the Columns
  - Set all four columns' Pref Width properties to USE_COMPUTED_SIZE so that the columns are only as wide as their content
  - For the leftmost column, set the Halignment property to RIGHT
  - For the rightmost column, set the Halignment property to CENTER.
- Step 10: Configuring the GridPane
  - Set the GridPane's Pref Width and Pref Height properties to USE_COMPUTED_SIZE so that it sizes itself, based on its contents

Building the GUI (cont.)

- Step 11: Specifying the Controller Class's Name
  - To ensure that an object of the controller class is created when the app loads the FXML file at runtime, specify ColorChooserController as the controller class's name in the FXML file as you've done previously.
- Step 12: Generating a Sample Controller Class
  - Select View > Show Sample Controller Skeleton, then copy this code into a ColorChooserController.java file and store the file in the same folder as ColorChooser.fxml

ColorChooser Main Application

```
1  // Fig. 13.8: ColorChooser.java
2  // Main application class that loads and displays the ColorChooser's GUI.
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  public class ColorChooser extends Application {
10    @Override
11    public void start(Stage stage) throws Exception {
12       Parent root =
13          FXMLLoader.load(getClass().getResource("ColorChooser.fxml"));
14
15       Scene scene = new Scene(root);
16       stage.setTitle("Color Chooser");
17       stage.setScene(scene);
18       stage.show();
19    }
20
21    public static void main(String[] args) {
22       launch(args);
23    }
24 }
```


ColorChooserController

```
1  // Fig. 13.9: ColorChooserController.java
2  // Controller for the ColorChooser app
3  import javafx.beans.value.ChangeListener;
4  import javafx.beans.value.ObservableValue;
5  import javafx.fxml.FXML;
6  import javafx.scene.control.Slider;
7  import javafx.scene.control.TextField;
8  import javafx.scene.paint.Color;
9  import javafx.scene.shape.Rectangle;
10
11 public class ColorChooserController {
12    // instance variables for interacting with GUI components
13    @FXML private Slider redSlider;
14    @FXML private Slider greenSlider;
15    @FXML private Slider blueSlider;
16    @FXML private Slider alphaSlider;
17    @FXML private TextField redTextField;
18    @FXML private TextField greenTextField;
19    @FXML private TextField blueTextField;
20    @FXML private TextField alphaTextField;
21    @FXML private Rectangle colorRectangle;
22
```


ColorChooserController (cont.)

```
23    // instance variables for managing
24    private int red = 0;
25    private int green = 0;
26    private int blue = 0;
27    private double alpha = 1.0;
28
29    public void initialize() {
30       // bind TextField values to corresponding Slider values
31       redTextField.textProperty().bind(
32          redSlider.valueProperty().asString("%.0f"));
33       greenTextField.textProperty().bind(
34          greenSlider.valueProperty().asString("%.0f"));
35       blueTextField.textProperty().bind(
36          blueSlider.valueProperty().asString("%.0f"));
37       alphaTextField.textProperty().bind(
38          alphaSlider.valueProperty().asString("%.2f"));
39
```

ColorChooserController (cont.)

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


ColorChooserController (cont.)

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


ColorChooserController Class (cont.)

- Property-to-Property Bindings
- Lines 31–38 set up property bindings between a Slider's value and the corresponding Text-Field's text so that changing a Slider updates the corresponding TextField
- Consider lines 31–32, which bind the redSlider's valueProperty to the redText-Field's textProperty:
  - redTextField.textProperty().bind(redSlider.valueProperty().asString("%.0f"));

ColorChooserController Class (cont.)

Property-to-Property Bindings (cont.)

- Each `TextField` has a `text` property that's returned by its `textProperty` method as a `StringProperty` (package `javafx.beans.property`)
- `StringProperty` method `bind` receives an `ObservableValue` as an argument
- When the `ObservableValue` changes, the bound property updates accordingly
- In this case the `ObservableValue` is the result of the expression `redSlider.valueProperty().asString("%.0f")`

`ColorChooserController` Class (cont.)

Property-to-Property Bindings (cont.)

- `Slider`'s `valueProperty` method returns the `Slider`'s `value` property as a `DoubleProperty`—an observable `double` value
- Because the `TextField`'s `text` property must be bound to a `String`, we call `DoubleProperty` method `asString`, which returns a `StringBinding` object (an `ObservableValue`) that produces a `String` representation of the `DoubleProperty`
- This version of `asString` receives a format-control `String` specifying the `DoubleProperty`'s format

`ColorChooserController` Class (cont.)

Property Listeners

- To perform an arbitrary task when a property's value changes, register a property listener
- Lines 41–80 register property listeners for the `Sliders`' `value` properties
- Consider lines 41–50, which register the `ChangeListener` that executes when the user moves the `redSlider`'s thumb
- Each `ChangeListener` stores the `int` value of the `newValue` parameter in a corresponding instance variable, then calls the `colorRectangle`'s `setFill` method to change its color, using `Color` method `rgb` to create the new `Color` object

Outline

1. JavaFX Overview
2. MVC Architectural Pattern
3. JavaFX Window Structure
4. Binding Properties
5. Data-Driven UIs

Data-Driven GUIs with JavaFX Collections

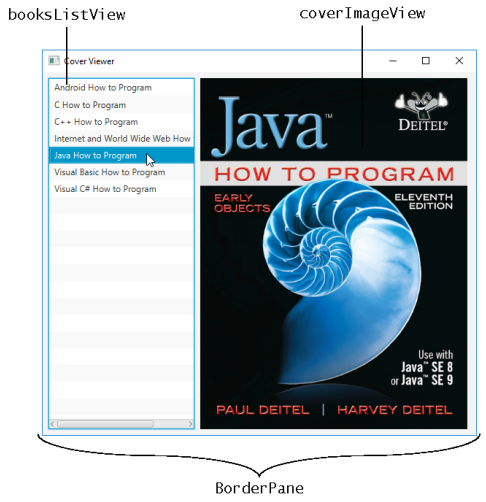
- JavaFX provides a comprehensive model for allowing GUIs to interact with data, e.g. edit and display data
- `FXCollections`: Creating and manipulating observable collections
- Example: List
  - `ListView`: UI component
  - `ObservableList`: Can be observed by corresponding UI components, e.g. ListView
    - If you make changes to an `ObservableList`, its observer will automatically be notified of those changes

Cover Viewer application

- Binds a list of Book objects to a ListView
- When the user selects an item in the ListView, the corresponding Book's cover image is displayed in an ImageView.

ListView and ObservableList

- ListView control: Display a collection of book titles
- Bind an ObservableList object to the ListView
  - If you make changes to an ObservableList, its observer (the ListView in this app) will automatically be notified of those changes
- Property listener to display the correct image when the user selects an item from the ListView—in this case, the property that changes is the selected item



Building the GUI (cont.)

Adding and Configuring the Controls

- Using the techniques you learned previously, create a `BorderPane`
- In the left area, place a `ListView` control, and in the center area, place an `ImageView` control
- 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`

Building the GUI (cont.)

Adding and Configuring the Controls (cont.)

- For the `ImageView`, set the Fit Width and Fit Height properties to `370` and `480`, respectively
- To size the `BorderPane` based on its contents, set its Pref Width and Pref Height to `USE_COMPUTED_SIZE`
- Also, set the Padding property to `8` to inset the `BorderPane` from the stage.

78

Building the GUI (cont.)

Specifying the Controller Class's Name

- To ensure that an object of the controller class is created when the app loads the FXML file at runtime, specify `CoverViewerController` as the controller class's name in the FXML file as you've done previously.

Generating a Sample Controller Class

- Select View > Show Sample Controller Skeleton, then copy this code into a `CoverViewerController.java` file and store the file in the same folder as `CoverViewer.fxml`

79

```
1  // Fig. 13.13: CoverViewer.java
2  // Main application class that loads and displays the CoverViewer's GUI.
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  public class CoverViewer extends Application {
10    @Override
11    public void start(Stage stage) throws Exception {
12       Parent root =
13          FXMLLoader.load(getClass().getResource("CoverViewer.fxml"));
14
15       Scene scene = new Scene(root);
16       stage.setTitle("Cover Viewer");
17       stage.setScene(scene);
18       stage.show();
19    }
20
21    public static void main(String[] args) {
22       launch(args);
23    }
24 }
```

80

```
1  // Fig. 13.14: CoverViewerController.java
2  // Controller for Cover Viewer application
3  import javafx.beans.value.ChangeListener;
4  import javafx.beans.value.ObservableValue;
5  import javafx.collections.FXCollections;
6  import javafx.collections.ObservableList;
7  import javafx.fxml.FXML;
8  import javafx.scene.control.ListView;
9  import javafx.scene.image.Image;
10 import javafx.scene.image.ImageView;
11
12 public class CoverViewerController {
13    // instance variables for interacting with GUI
14    @FXML private ListView<Book> booksListView;
15    @FXML private ImageView coverImageView;
16
17    // stores the list of Book Objects
18    private final ObservableList<Book> books =
19       FXCollections.observableArrayList();
20
```

81

```
21     // initialize controller
22     public void initialize() {
23        // populate the ObservableList<Book>
24        books.add(new Book("Android How to Program",
25           "/images/small/androidhtp.jpg", "/images/large/androidhtp.jpg"));
26        books.add(new Book("C How to Program",
27           "/images/small/chtp.jpg", "/images/large/chtp.jpg"));
28        books.add(new Book("C++ How to Program",
29           "/images/small/cpphtp.jpg", "/images/large/cpphtp.jpg"));
30        books.add(new Book("Internet and World Wide Web How to Program",
31           "/images/small/iw3htp.jpg", "/images/large/iw3htp.jpg"));
32        books.add(new Book("Java How to Program",
33           "/images/small/jhtp.jpg", "/images/large/jhtp.jpg"));
34        books.add(new Book("Visual Basic How to Program",
35           "/images/small/vbhtp.jpg", "/images/large/vbhtp.jpg"));
36        books.add(new Book("Visual C# How to Program",
37           "/images/small/vcshtp.jpg", "/images/large/vcshtp.jpg"));
38        booksListView.setItems(books); // bind booksListView to books
39
40        // when ListView selection changes, show large cover in ImageView
41        booksListView.getSelectionModel().selectedItemProperty().
42           addListener(
43              new ChangeListener<Book>() {
44                 @Override
45                 public void changed(ObservableValue<? extends Book> ov,
46                    Book oldValue, Book newValue) {
47                    coverImageView.setImage(
48                       new Image(newValue.getLargeImage()));
49                 }
50              }
51           );
52     }
53  }
```


CoverViewerController Class (cont.)

@FXML Instance Variables

- Lines 14–15 declare the controller's `@FXML` instance variables
- In this case, the `ListView` displays `Book` objects
- Class `Book` contains three `String` instance variables with *set/get* methods:
  - `title`—the book's title.
  - `thumbImage`—the path to the book's thumbnail image (used in the next example).
  - `largeImage`—the path to the book's large cover image
- The class also provides a `toString` method that returns the `Book`'s title and a constructor that initializes the three instance variables
- Copy `Book.java` from this chapter's examples folder into the folder that contains `CoverViewer.fxml`, `CoverViewer.java` and `CoverViewerController.java`

CoverViewerController Class (cont.)

Instance Variable books

- Lines 18–19 define the `books` instance variable as an `ObservableList<Book>` and initialize it by calling `FXCollections` `static` method `observableArrayList`
- This method returns an empty collection object (similar to an `ArrayList`) that implements the `Observable-List` interface

CoverViewerController Class (cont.)

Initializing the books `ObservableList`

- Lines 24–37 in method `initialize` create and add `Book` objects to the `books` collection
- Line 38 passes this collection to `ListView` method `setItems`, which binds the `ListView` to the `ObservableList`
- This *data binding* allows the `ListView` to display the `Book` objects automatically
- By default, the `ListView` displays each `Book`'s `String` representation

CoverViewerController Class (cont.)

Listening for `ListView` Selection Changes

- To synchronize the book cover that's being displayed with the currently selected book, we listen for changes to the `ListView`'s selected item
- By default a `ListView` supports single selection
  - `ListView`s also support multiple selection
- Selection is managed by the `ListView`'s `MultipleSelectionModel` (a subclass of `SelectionModel` from package `javafx.scene.control`
  - Contains observable properties and various methods for manipulating the corresponding `ListView`'s items
- To respond to selection changes, you register a listener for the `MultipleSelectionModel`'s `selectedItem` property (lines 41–51)

86

CoverViewerController Class (cont.)

Listening for `ListView` Selection Changes (cont.)

- `ListView` method `getSelectionModel` returns a `MultipleSelectionModel` object
- In this example, `MultipleSelectionModel`'s `selectedItemProperty` method returns a `ReadOnlyObjectProperty<Book>`
- Corresponding `ChangeListener` receives as `oldValue` and `newValue` the previously selected and newly selected `Book` objects, respectively
- Lines 47–48 use `newValue`'s large image path to initialize a new `Image` (package `javafx.scene.image`)—this loads the image from that path
- We then pass the new `Image` to the `coverImageView`'s `setImage` method to display the `Image`

87

More examples/tutorials

- https://o7planning.org/11009/javafx
- https://code.makery.ch/library/javafx-tutorial/

88