Chapter 14 JavaFX Basics

Motivations

JavaFX is a new framework for developing Java GUI programs. The JavaFX API is an excellent example of how the object-oriented principle is applied. This chapter serves two purposes. First, it presents the basics of JavaFX programming. Second, it uses JavaFX to demonstrate OOP. Specifically, this chapter introduces the framework of JavaFX and discusses JavaFX GUI components and their relationships.
Objectives

❑ To distinguish between JavaFX, Swing, and AWT (§14.2).
❑ To write a simple JavaFX program and understand the relationship among stages, scenes, and nodes (§14.3).
❑ To create user interfaces using panes, UI controls, and shapes (§14.4).
❑ To use binding properties to synchronize property values (§14.5).
❑ To use the common properties style and rotate for nodes (§14.6).
❑ To create colors using the Color class (§14.7).
❑ To create fonts using the Font class (§14.8).
❑ To create images using the Image class and to create image views using the ImageView class (§14.9).
❑ To layout nodes using Pane, StackPane, FlowPane, GridPane, BorderPane, HBox, and VBox (§14.10).
❑ To display text using the Text class and create shapes using Line, Circle, Rectangle, Ellipse, Arc, Polygon, and Polyline (§14.11).
❑ To develop the reusable GUI components ClockPane for displaying an analog clock (§14.12).

JavaFX vs. Swing & AWT

- Initially Java came with the AWT (Abstract Windows Toolkit) GUI (Graphical User Interface) library. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. AWT components are platform-dependent

- AWT was replaced by a more robust, versatile, and flexible library, Swing, in 1998. Swing components are platform-independent

- Java 8 introduced in 2014 the JavaFX GUI framework for developing Rich Internet Applications (RIA) that provide a desktop-like experience on Web applications. JavaFX is an excellent example of how the object-oriented principles are applied. JavaFX components are platform-independent
Structure of a JavaFX Program

Every JavaFX program is defined in a class that extends javafx.application.Application.

1. Extend Application
2. Override start(Stage)
3. Create Nodes (e.g., Button)
4. Place the Nodes in the Scene
5. Place the Scene on Stage
6. Show Stage

Panels, UI Controls, and Shapes

1. Extend Application
2. Override start(Stage)
3. Create Nodes
4. Place Nodes in a Parent
5. Place the Parent in the Scene
6. Place the Scene on Stage
7. Show Stage

Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.
Example: Circle at the Center of a Pane

 Property Binding

- Property binding enables a property of a target object to be bound to a property of the source object. If the value of the property in the source object changes, the corresponding target property changes automatically.

- The target object is called a binding object or a binding property. The source object is called a bind-able or observable object.

- Syntax: `target.bind(source);`

```java
circle.centerXProperty().bind(pane.widthProperty().divide(2));
circle.centerYProperty().bind(pane.heightProperty().divide(2));
```

The center of the circle will always remain in the middle of the pane.
Property Binding: **Accessor, Mutator & Property Accessor**

- Each binding property (e.g., centerX) in a JavaFX class (e.g., Circle) has the usual accessor (e.g., getCenterX()) and a mutator (e.g., setCenterX(double))

- Each binding property also has a property accessor for returning the property itself. The name for this property accessor method should be the property name followed by the word Property

```
public class SomeClassName {
    private PropertyType x;
    /** Value getter method */
    public PropertyType getValue() { ... }
    /** Value setter method */
    public void setValue(PropertyType value) { ... }
    /** Property accessor */
    public PropertyType xProperty() { ... }
}
```

```
public class Circle {
    private DoubleProperty centerX;
    /** Value getter method */
    public double getCenterX() { ... }
    /** Value setter method */
    public void setCenterX(double value) { ... }
    /** Property accessor */
    public DoubleProperty centerXP() { ... }
}
```

 Uni- and Bi-directional Binding

- The binding demonstrated in the ShowCircleCentered example is known as **unidirectional binding**

- Occasionally, it is useful to synchronize two properties so that a change in one property is reflected in another object, and vice versa. This is called a **bidirectional binding**

- If the target and source are both binding properties and observable properties, they can be bound bi-directionally using `bindBidirectional()`
Node Properties: *style* & *rotate*

- The abstract *Node* class defines many properties & methods. Here we focus upon two of those properties: *style* and *rotate*

- Syntax for setting the *style* of a node is *styleName:value*. A style property is defined with a prefix `-fx-`

- Multiple *style* properties for a node can be set together separated by semicolon. For example, the statement:
  
  ```
  circle.setStyle("-fx-stroke: black; -fx-fill: red;"进修);
  ```

  sets two style properties for a `circle`

- The *rotate* property is used to specify an angle in degrees for rotating the node. For example, the following code rotates a button by 80 degrees:
  
  ```java
  button.setRotate(80);
  ```

The Color Class

<table>
<thead>
<tr>
<th>javafx.scene.paint.Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>-red: double</td>
</tr>
<tr>
<td>-green: double</td>
</tr>
<tr>
<td>-blue: double</td>
</tr>
<tr>
<td>-opacity: double</td>
</tr>
<tr>
<td>+Color(r: double, g: double, b: double, opacity: double)</td>
</tr>
<tr>
<td>+brighter(): Color</td>
</tr>
<tr>
<td>+darker(): Color</td>
</tr>
<tr>
<td>+color(r: double, g: double, b: double): Color</td>
</tr>
<tr>
<td>+color(r: double, g: double, b: double, opacity: double): Color</td>
</tr>
<tr>
<td>+rgb(r: int, g: int, b: int): Color</td>
</tr>
<tr>
<td>+rgb(r: int, g: int, b: int, opacity: double): Color</td>
</tr>
</tbody>
</table>

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

- The red value of this Color (between 0.0 and 1.0).
- The green value of this Color (between 0.0 and 1.0).
- The blue value of this Color (between 0.0 and 1.0).
- The opacity of this Color (between 0.0 and 1.0).

  *Creates a Color* with the specified red, green, blue, and opacity values.

  *Creates a Color* that is a brighter version of this Color.

  *Creates a Color* that is a darker version of this Color.

  *Creates an opaque Color* with the specified red, green, and blue values.

  *Creates a Color* with the specified red, green, blue, and opacity values.

  *Creates a Color* with the specified red, green, blue, and blue values in the range from 0 to 255.

  *Creates a Color* with the specified red, green, blue, and blue values in the range from 0 to 255 and a given opacity.
The **Font Class**

```java
javafx.scene.text.Font
-size: double
-name: String
-family: String
+Font(size: double)
+Font(name: String, size: double)
+Font(name: String, size: FontWeight)
+Font(name: String, w: FontWeight, p: FontPosture, size: double)
+getFamilies(): List<String>
+getFontNames(): List<String>
```

The size of this font.
The name of this font.
The family of this font.
Creates a Font with the specified size.
Creates a Font with the specified full font name and size.
Creates a Font with the specified name and size.
Creates a Font with the specified name, weight, and size.
Creates a Font with the specified name, weight, posture, and size.
Returns a list of font family names.
Returns a list of full font names including family and weight.

The **Image & ImageView Classes**

- The **Image** class represents a graphical image and is used for loading an image from a specified filename or URL. For example:
  - `new Image("image/us.gif")` creates an Image object for the file `us.gif` under the directory `image` in the Java class directory
  - `new Image("http://www.armstrong.edu/image/us.gif")` creates an Image object for the image file in the URL on the Web

- **ImageView** is used for displaying an Image object. For example,
  ```java
  Image image = new Image("image/us.gif");
  ImageView imageView = new ImageView(image);
  ```
Pane and its Subclasses

- Pane (or any of its subclasses) is used for organizing nodes
- The Pane is then placed in a Scene, which is placed on a Stage
- The Stage is then revealed on the screen using show()
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 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.

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

- `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`
+ `setAlignment(child: Node, value: Alignment): void`
+ `setAlignment(child: Node, value: VPos): void`

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.

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.
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.

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

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

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.

- `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

JavaFX provides many shape classes for drawing text, lines, rectangles, circles, 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.

```java
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.

---

**Line**

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.

```java
javafx.scene.shape.Line

-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty

+Line()
+Line(startX: double, startY: double, endX: double, endY: double)
```

- The x-coordinate of the start point.
- The y-coordinate of the start point.
- The x-coordinate of the end point.
- The y-coordinate of the end point.
- Creates an empty Line.
- Creates a Line with the specified starting and ending points.
**Rectangle**

The x-coordinate of the upper-left corner of the rectangle (default 0).
The y-coordinate of the upper-left corner of the rectangle (default 0).
The width of the rectangle (default: 0).
The height of the rectangle (default: 0).
The arcWidth of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).
The arcHeight of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

Creates an empty `Rectangle`.
Creates a `Rectangle` with the specified upper-left corner point, width, and height.

**Circle**

The x-coordinate of the center of the circle (default 0).
The y-coordinate of the center of the circle (default 0).
The radius of the circle (default: 0).

Creates an empty `Circle`.
Creates a `Circle` with the specified center.
Creates a `Circle` with the specified center and radius.
**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.

```java
javax.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)
```

**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.

```java
javax.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.
**Polygon & Polyline**

(a) Polygon

(b) Polyline

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

The accessor and mutator methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

**Case Study: The ClockPane Class**

ClockPane displays the time on a static analog clock

The hour in the clock.
The minute in the clock.
The second in the clock.

Constructs a default clock for the current time.
Constructs a clock with the specified time.

Sets hour, minute, and second for current time.
Sets clock pane’s width and repaint the clock.
Sets clock pane’s height and repaint the clock,
public class DisplayClock extends Application {
    public void start(Stage primaryStage) {
        ClockPane clock = new ClockPane();
        String timeString = clock.getHour() + ":" + clock.getMinute() + ":" + clock.getSecond();
        Label lblTime = new Label(timeString);
        BorderPane pane = new BorderPane();
        pane.setCenter(clock); pane.setBottom(lblCurrentTime);
        BorderPane.setAlignment(lblTime, Pos.TOP_CENTER);
        Scene scene = new Scene(pane, 250, 250);
        primaryStage.setTitle("DisplayClock");
        primaryStage.setScene(scene); primaryStage.show();
    }
}

What info about ClockPane is needed to write DisplayClock?

All needed info is available on the previous slide. We don’t need to see the code for ClockPane to write the DisplayClock class. This means that ClockPane is a well

clock: The Details

- ClockPane displays a clock for the current time or for a time
  specified in terms of hour, minute and second
- The current hour, minute & second is obtained using the
  get(Calendar.HOUR), get(Calendar.MINUTE) and
  get(Calendar.SECOND) methods of GregorianCalendar
- The w & h properties represent the width & height of the Pane
- paintClock() places eight items in the Pane using addAll():
  - Circle at the center of the pane with a radius proportional to w & h
  - Text shapes for the hours 12, 3, 6, 9
  - Line shapes for second, minute and hour hands
- paintClock() is invoked whenever a new property (w, h, hour, minute, second) is set. The old contents of the Pane are removed using clear() before any update
Second Midterm Exam

- 60 minutes duration
- Will cover all lectures delivered before the exam date
- Will consist of MCQ’s and coding/programming tasks
- If you miss this exam for *any valid* reason, you will have to appear for an exam after the approval of your absence by the vice-dean of CCSE.