MC322 - Object Oriented Programming
Lesson 11.1
Graphical Interface - JavaFX Basics

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP

JavaFX Framework Overview

- JavaFX is a new framework for developing Java GUI programs.
- The JavaFX API exemplifies the application of object-oriented principles.

Chapter Purposes

- Present the basics of JavaFX programming.
- Demonstrate object-oriented design and programming using JavaFX.

- Introduce the framework of JavaFX.
- Discuss JavaFX GUI components and their relationships.
- Learn to develop simple GUI programs using: Layout panes, Buttons, Labels, Text fields, Colors, Fonts, Images, Image views, Shapes

Evolution of Java GUI Frameworks

- Java initially introduced GUI classes in the Abstract Windows Toolkit (AWT).
  - AWT is suitable for simple GUIs but not comprehensive projects.
  - Prone to platform-specific bugs.
- AWT was replaced by Swing components.
  - Swing is robust, versatile, and flexible.
  - Painted directly on canvases using Java code.
  - Depend less on the target platform, using fewer native GUI resources.
  - Designed for desktop GUI applications.
- Swing will not receive further enhancements, making it obsolete.
- Swing is now replaced by JavaFX.
  - JavaFX incorporates modern GUI technologies for rich Internet applications (RIAs).
  - JavaFX applications run seamlessly on desktops and web browsers.
  - Provides multi-touch support for touch-enabled devices like tablets and smartphones.
  - Built-in support for 2D, 3D, animation, video, and audio playback.
  - Can run as stand-alone applications or from a browser.
- JavaFX is simpler to learn and use for new Java programmers.
- JavaFX is the new tool for developing cross-platform rich Internet applications on desktops, hand-held devices, and the Web.

The Basic Structure of a JavaFX Program

Basic Structure of a JavaFX Program

- `MyJavaFX.java` program illustrates the basic structure.
- Every JavaFX program is defined in a class that extends `javafx.application.Application`.
- The `launch` method (line 22) is a static method defined in the Application class for launching a stand-alone JavaFX application.
- The `main` method (lines 21–23) may be needed to launch a JavaFX program from an IDE with limited JavaFX support.
- The main class overrides the `start` method defined in `javafx.application.Application` (line 8).
- JVM constructs an instance of the class using its no-arg constructor and invokes its `start` method.
- The `start` method places UI controls in a scene and displays the scene in a stage.
- Line 10 creates a `Button` object and places it in a `Scene` object (line 11).
- A `Scene` object can be created using the constructor `Scene(node, width, height)`.
- A `Stage` object is a window, with the primary stage created by the JVM when the application is launched.

Working with Multiple Stages in JavaFX

- You can create additional stages if needed.
- The JavaFX program in `MultipleStageDemo` displays two stages.
- The `main` method is omitted in the listing since it is identical for every JavaFX application.
- From now on, the `main` method will not be listed in our JavaFX source code for brevity.
- By default, the user can resize the stage.
- To prevent the user from resizing the stage, invoke `stage.setResizable(false)`.

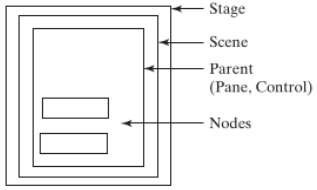
Panes, UI Controls, and Shapes

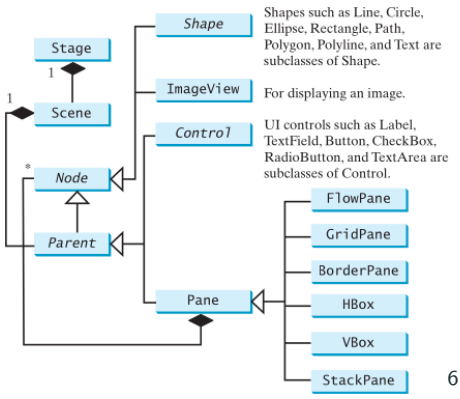
Displaying and Laying Out Nodes in JavaFX

- When you run `MyJavaFX` the button is always centered in the scene and occupies the entire window no matter how you resize it.
- Fix the problem by setting a button's position and size properties.
- A better approach is to use container classes, called panes, for automatically laying out the nodes in a desired location and size.
- Place nodes inside a pane and then place the pane into a scene.
- A node is a visual component such as:
  - Shape (e.g., text, line, circle, ellipse, rectangle, arc, polygon, polyline)
  - Image view
  - UI control (e.g., label, button, check box, radio button, text field, text area)
  - Pane

Basic Structure of a JavaFX Program

- A scene can be displayed in a stage, as shown in Figure.
- The relationship among Stage, Scene, Node, Control, and Pane is illustrated in the UML diagram, as shown in Figure.
- A Scene can contain a Control or a Pane, but not a Shape or an ImageView.
- A Pane can contain any subtype of Node.
- Create a Scene using the constructor `Scene(Parent, width, height)` or `Scene(Parent)`.
- The dimension of the scene is automatically decided in the latter constructor.
- Every subclass of Node has a no-arg constructor for creating a default node.





Using StackPane in JavaFX

- The program **ButtonInPane.java** creates a `StackPane` (line 11) and adds a button as a child of the pane (line 12).
- The `getChildren()` method returns an instance of `javafx.collections.ObservableList`.
- `ObservableList` behaves very much like an `ArrayList` for storing a collection of elements.
- Invoking `add(e)` adds an element to the list.
- The `StackPane` places the nodes (the Button) in the center of the pane on top of each other.
- Here, there is only one node in the pane.
- The `StackPane` respects a node's preferred size.
- So you see the button displayed in its preferred size.

Creating and Displaying a Circle in JavaFX

- **ShowCircle.java** creates a `Circle` (line 12) and sets its center at (100, 100) (lines 13–14).
- The scene is created with a width and height of 200 (line 24), making (100, 100) the center of the scene.
- The radius of the circle is set to 50 (line 15).
- Measurement units for graphics in Java are in pixels.
- The stroke color (color to draw the circle) is set to black (line 16).
- The fill color (color to fill the circle) is set to white (line 17).
- Set the color to `null` to specify that no color is set.
- The program creates a `Pane` (line 20) and places the circle in the pane (line 21).
- Coordinates of the upper left corner of the pane is (0, 0) in the Java coordinate system.
- The x-coordinate increases from left to right and the y-coordinate increases downward in the Java coordinate system.
- The pane is placed in the scene (line 24) and the scene is set in the stage (line 26).
- The circle is displayed in the center of the stage.
- Resizing the window does not center the circle.
- To center the circle as the window resizes, reset the x- and y-coordinates of the circle center to the center of the pane.
- This can be done using property binding, introduced in the next section.

Property Binding

Property Binding in JavaFX

- JavaFX introduces property binding to enable a target object to be bound to a source object.
- If the value in the source object changes, the target object changes automatically.
- The target object is called a binding object or binding property.
- The source object is called a bindable object or observable object.

- In the previous example, the circle is not centered after the window is resized.
- To display the circle centered as the window resizes, reset the x- and y-coordinates of the circle center to the center of the pane.
- This can be done by binding the `centerX` with pane's `width/2` and 1ttcenterY with pane's `height/2`, as shown in **ShowCircleCentered.java**

Binding Properties in JavaFX

- The `Circle` class has the `centerX` property for representing the x-coordinate of the circle center.
- This property, like many properties in JavaFX classes, can be used both as a target and source in a property binding.
- A target listens to changes in the source and automatically updates itself once a change is made in the source.
- A target binds with a source using the `bind` method:
  - `target.bind(source);`
- The `bind` method is defined in the `javafx.beans.property.Property` interface.
- A binding property is an instance of `javafx.beans.property.Property`.
- A source object is an instance of the `javafx.beans.value.ObservableValue` interface.
- An `ObservableValue` is an entity that wraps a value and allows observing the value for changes.

Binding Properties in JavaFX

- JavaFX defines binding properties for primitive types and strings:
  - For a double/float/long/int/boolean value: `DoubleProperty`, `FloatProperty`, `LongProperty`, `IntegerProperty`, `BooleanProperty`.
  - For a string: `StringProperty`.
- These properties are also subtypes of `ObservableValue`, so they can be used as source objects for binding properties.

Binding Property Conventions in JavaFX

- Each binding property (e.g., `centerX`) in a JavaFX class (e.g., `Circle`) has:
  - A getter method for returning the property's value (e.g., `getCenterX()`).
  - A setter method for setting the property's value (e.g., `setCenterX(double)`).
  - A getter method for returning the property itself, named by appending "Property" to the property name (e.g., `centerXProperty()`).
- Naming conventions:
  - `getCenterX()` is the value getter method.
  - `setCenterX(double)` is the value setter method.
  - `centerXProperty()` is the property getter method.

Example: Binding Property Implementation

SomeClassName shows the convention for defining a binding property in a class.

```
public class SomeClassName {
    private PropertyType x;
    /** Value getter method */
    public propertyValueType getX() { ... }

    /** Value setter method */
    public void setX(propertyValueType
        value) { ... }

    /** Property getter method */
    public PropertyType xProperty() { ... }
}
```

Circle shows a concrete example where `centerX` is a binding property of the type `DoubleProperty`.

```
public class Circle {
    private DoubleProperty centerX;
    public double getCenterX() { ... }

    /** Value setter method */
    public void setCenterX(double value) {
        ... }

    /** Property getter method */
    public DoubleProperty centerXProperty()
        { ... }
}
```

Binding Circle Properties to Pane Dimensions

- ShowCircleCentered.java is the same as ShowCircle.java except that it binds circle's `centerX` and `centerY` properties to half of pane's width and height (lines 16–17).
- `circle.centerXProperty()` returns `centerX` and `pane.widthProperty()` returns `width`.
- Both `centerX` and `width` are binding properties of the `DoubleProperty` type.
- Numeric binding property classes like `DoubleProperty` and `IntegerProperty` contain `add`, `subtract`, `multiply`, and `divide` methods.
- These methods allow operations on a value in a binding property and return a new observable property.
- `pane.widthProperty().divide(2)` returns a new observable property representing half of the pane's width.

- The statement:

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

- is the same as:

```
centerX.bind(width.divide(2));
```

- Since `centerX` is bound to `width.divide(2)`, when the pane's width changes, `centerX` automatically updates to match `pane's width / 2`.

Creating and Binding DoubleProperty Instances

- The program creates an instance of `DoubleProperty` using `SimpleDoubleProperty(1)` (line 6).
- Note that `DoubleProperty`, `FloatProperty`, `LongProperty`, `IntegerProperty`, and `BooleanProperty` are abstract classes.
- Their concrete subclasses `SimpleDoubleProperty`, `SimpleFloatProperty`, `SimpleLongProperty`, `SimpleIntegerProperty`, and `SimpleBooleanProperty` are used to create instances of these properties.
- These classes are similar to wrapper classes `Double`, `Float`, `Long`, `Integer`, and `Boolean`, with additional features for binding to a source object.
- The program binds `d1` with `d2` (line 8). Now the values in `d1` and `d2` are the same.
- After setting `d2` to 70.2 (line 11), `d1` also becomes 70.2 (line 13).
- The binding demonstrated in this 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 bidirectional binding.
- If the target and source are both binding properties and observable properties, they can be bound bidirectionally using the `bindBidirectional` method.

Common Properties and Methods for Nodes

Common Node Properties: Style and Rotate

Style Properties

- JavaFX style properties are similar to CSS used in HTML.
- In JavaFX, a style property is defined with a prefix `-fx-`.
- Each node has its own style properties.
- Syntax for setting a style: `styleName:value`.
- Multiple style properties can be set together separated by semicolon (;).
- Example:

```
circle.setStyle("-fx-stroke: black; -fx-fill:
    red;");
```

- Equivalent to:

```
circle.setStroke(Color.BLACK);
circle.setFill(Color.RED);
```

- Incorrect JavaFX CSS will be ignored but the program will still compile and run.

Rotate Property

- The rotate property specifies an angle in degrees for rotating the node from its center.
- Positive degree: rotation clockwise.
- Negative degree: rotation counterclockwise.
- Example:

```
button.setRotate(80);
```

Example: Applying Style and Rotate Properties

- **NodeStyleRotateDemo.java** gives an example that creates a button, sets its style, and adds it to a pane.
- It then rotates the pane 45 degrees and sets its style with a border color red and a background color light gray, as shown in Figure 14.8.

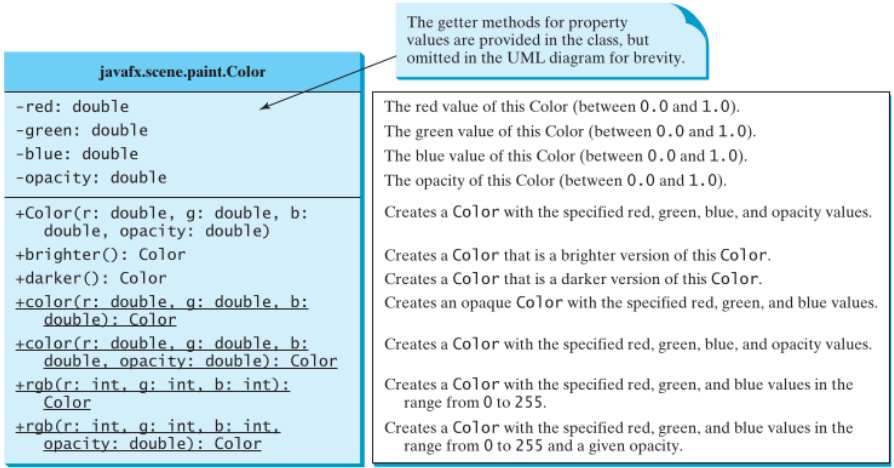
```
Button button = new Button("OK");
button.setStyle("-fx-stroke: black; -fx-fill: red;");

Pane pane = new Pane();
pane.getChildren().add(button);
pane.setRotate(45);
pane.setStyle("-fx-border-color: red; -fx-background-color: lightgray;");
```

- The `Node` class contains many useful methods that can be applied to all nodes.
- For example, you can use the `contains(double x, double y)` method to test whether a point (x, y) is inside the boundary of a node.

The Color Class

Painting Nodes in JavaFX

- JavaFX defines the abstract `Paint` class for painting a node.
- The `javafx.scene.paint.Color` is a concrete subclass of `Paint`.
- `Color` is used to encapsulate colors, as shown in Figure 14.9.



Creating and Using Colors in JavaFX

- A color instance can be constructed using the following constructor:

```
public Color(double r, double g, double b, double opacity);
```

- `r`, `g`, and `b` specify a color by its red, green, and blue components with values in the range from 0.0 (darkest shade) to 1.0 (lightest shade).
- The opacity value defines the transparency of a color within the range from 0.0 (completely transparent) to 1.0 (completely opaque).
- This is known as the RGBA model, where RGBA stands for red, green, blue, and alpha. The alpha value indicates the opacity.
- Example:

```
Color color = new Color(0.25, 0.14, 0.333, 0.51);
```

- The `Color` class is immutable. Once a `Color` object is created, its properties cannot be changed.
- The `brighter()` method returns a new `Color` with larger red, green, and blue values.
- The `darker()` method returns a new `Color` with smaller red, green, and blue values. The opacity value remains the same as in the original `Color` object.

Creating and Using Colors in JavaFX

- You can also create a `Color` object using the static methods `color(r, g, b)`, `color(r, g, b, opacity)`, `rgb(r, g, b)`, and `rgb(r, g, b, opacity)`.
- Alternatively, use one of the many standard colors defined as constants in the `Color` class, such as `BEIGE`, `BLACK`, `BLUE`, `BROWN`, `CYAN`, `DARKGRAY`, `GOLD`, `GRAY`, `GREEN`, `LIGHTGRAY`, `MAGENTA`, `NAVY`, `ORANGE`, `PINK`, `RED`, `SILVER`, `WHITE`, and `YELLOW`.
- Example:

```
circle.setFill(Color.RED);
```

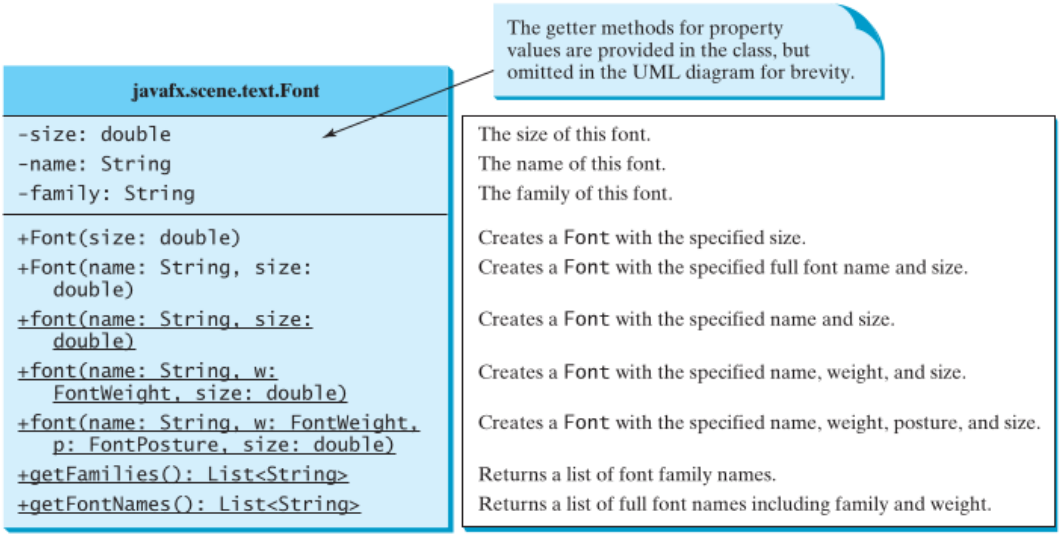
The Font Class

Creating and Using Fonts in JavaFX

- You can set fonts for rendering text using the `javafx.scene.text.Font` class.
- A `Font` instance can be constructed using its constructors or static methods.
- A `Font` is defined by its name, weight, posture, and size.
- Examples of font names: Times, Courier, and Arial.
- Obtain a list of available font family names by invoking the static `getFamilies()` method.
- `List` is an interface that defines common methods for a list. `ArrayList` is a concrete implementation of `List`.
- Font postures are constants: `FontPosture.ITALIC` and `FontPosture.REGULAR`.
- Example statements creating two fonts:

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 12);
```

Creating and Using Fonts in JavaFX



The Image and ImageView Classes

Using Images in JavaFX

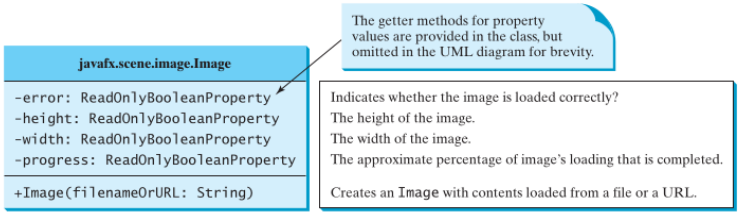
- The `Image` class represents a graphical image and the `ImageView` class can be used to display an image.
- The `javafx.scene.image.Image` class is used for loading an image from a specified filename or a URL.
- Examples:
  - `new Image("image/us.gif")` creates an `Image` object for the image file `us.gif` under the directory `image` in the Java class directory.
  - `new Image("http://www.cs.armstrong.edu/liang/image/us.gif")` creates an `Image` object for the image file in the URL on the Web.
- The `javafx.scene.image.ImageView` is a node for displaying an image.
- An `ImageView` can be created from an `Image` object.
- Example:

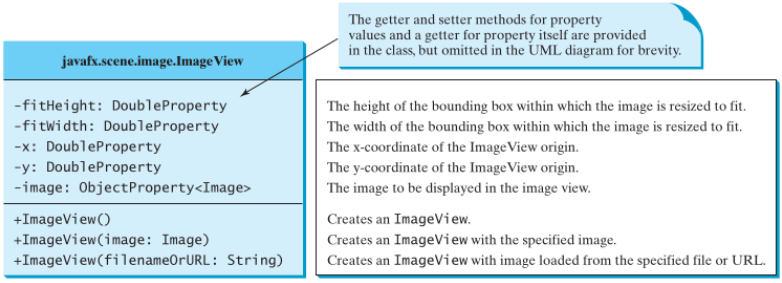
```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

- Alternatively, you can create an `ImageView` directly from a file or a URL:

```
ImageView imageView = new ImageView("image/us.gif");
```

Using Images in JavaFX





Using HBox and ImageView in JavaFX

- The program creates an `HBox` (line 14).
  - An `HBox` is a pane that places all nodes horizontally in one row.
- The program creates an `Image`, then an `ImageView` for displaying the image, and places the `ImageView` in the `HBox` (line 17).
- The program creates the second `ImageView` (line 19), sets its `fitHeight` and `fitWidth` properties (lines 20–21), and places the `ImageView` into the `HBox` (line 22).
- The program creates the third `ImageView` (line 24), rotates it 90 degrees (line 25), and places it into the `HBox` (line 26).
  - The `setRotate` method is defined in the `Node` class and can be used for any node.
- An `Image` object can be shared by multiple nodes. In this case, it is shared by three `ImageView` instances.
- However, a node such as `ImageView` cannot be shared. You cannot place an `ImageView` multiple times into a pane or scene.
- Note that you must place the image file in the same directory as the class file, as shown in the following figure.

Layout Panes

Layout Panes in JavaFX

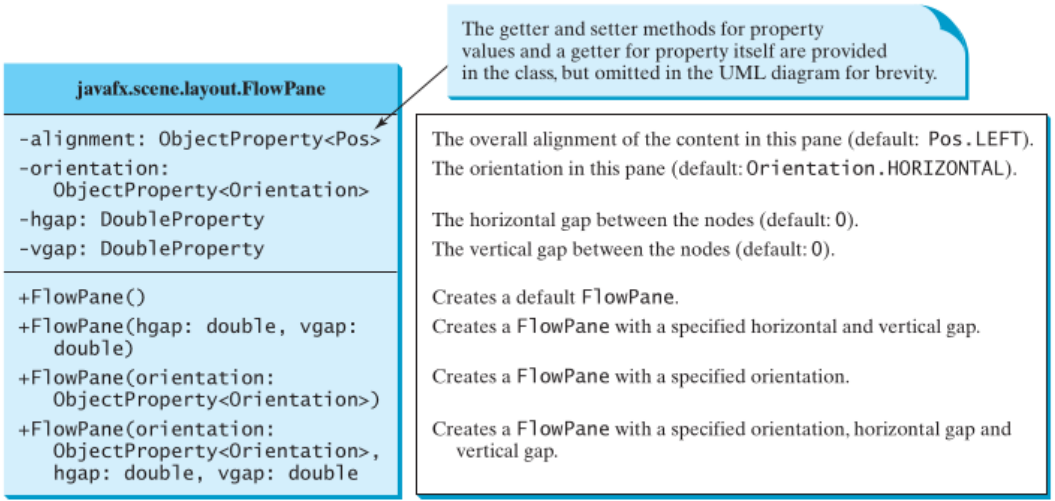
JavaFX provides many types of panes for organizing nodes in a container. You have used the layout panes Pane, StackPane, and HBox in the preceding sections for containing nodes.

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.

Using Pane and Specialized Panes in JavaFX

- You have used the `Pane` in `ShowCircle.java`.
  - A `Pane` is usually used as a canvas for displaying shapes.
  - `Pane` is the base class for all specialized panes.
- You have used a specialized pane `StackPane` in `ButtonInPane.java`.
  - Nodes are placed in the center of a `StackPane`.
- Each pane contains a list for holding nodes in the pane.
  - This list is an instance of `ObservableList`, which can be obtained using the pane's `getChildren()` method.
  - You can use the `add(node)` method to add an element to the list.
  - Use `addAll(node1, node2, ...)` to add a variable number of nodes to the pane.

Using Pane and Specialized Panes in JavaFX



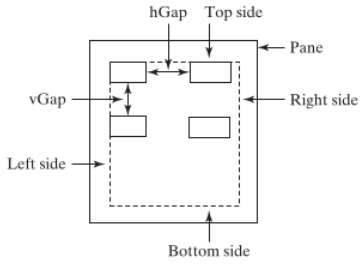
Using FlowPane in JavaFX

- **ShowFlowPane.java** creates a `FlowPane` (line 13) and sets its padding property with an `Insets` object (line 14).
- An `Insets` object specifies the size of the border of a pane.
- The constructor `Insets(11, 12, 13, 14)` creates an `Insets` with the border sizes for top (11), right (12), bottom (13), and left (14) in pixels, as shown in Figure.
- You can also use the constructor `Insets(value)` to create an `Insets` with the same value for all four sides.
- The `hgap` and `vgap` properties are set in lines 15–16 to specify the horizontal gap and vertical gap between two nodes in the pane, as shown in Figure.

- Example:

```
FlowPane pane = new FlowPane();
pane.setPadding(new Insets(11, 12,
    13, 14));
pane.setHgap(10);
pane.setVgap(10);
```

- This code sets the padding and gaps in the `FlowPane` to create spacing around and between the nodes.



Managing Nodes in FlowPane

- Each `FlowPane` contains an `ObservableList` for holding the nodes.
- This list can be obtained using the `getChildren()` method (line 19).
- Add a node to a `FlowPane` using the `add(node)` or `addAll(node1, node2, ...)` method.
- Remove a node using the `remove(node)` method or use the `removeAll()` method to remove all nodes from the pane.
- The program adds labels and text fields into the pane (lines 19–24).
- Example:

```
FlowPane pane = new FlowPane();
pane.getChildren().addAll(label1,
    textField1, label2, textField2);
```

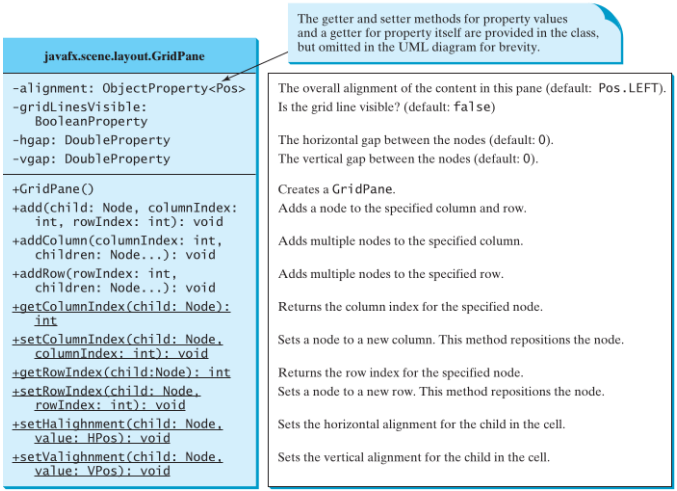
- The program adds the pane to the scene (line 27), sets the scene in the stage (line 29), and displays the stage (line 30).
- If you resize the window, the nodes are automatically rearranged to fit in the pane.
- In Figure 14.16a, the first row has three nodes, but in Figure 14.16b, the first row has four nodes due to increased width.
- Adding a node such as a text field to a pane multiple times or to different panes will cause a runtime error.
- Example:

```
pane.getChildren().add(tfMi);  // Correct
pane.getChildren().addAll(tfMi, tfMi);  //
    Incorrect, runtime error
```

GridPane

GridPane in JavaFX

A `GridPane` arranges nodes in a grid (matrix) formation. Nodes are placed in the specified column and row indices.



Using GridPane in JavaFX

- The program creates a `GridPane` (line 16) and sets its properties (lines 17–20).
- The alignment is set to the center position (line 17), which causes the nodes to be placed in the center of the grid pane.
- If you resize the window, the nodes remain in the center of the grid pane.
- The program adds the label in column 0 and row 0 (line 23).
- The column and row index starts from 0.
- The add method places a node in the specified column and row.
- Not every cell in the grid needs to be filled.
- Example: A button is placed in column 1 and row 3 (line 30), but there are no nodes placed in column 0 and row 3.

- To remove a node from a `GridPane`, use `pane.getChildren().remove(node)`.
- To remove all nodes, use `pane.getChildren().removeAll()`.
- Example:

```
gridPane.getChildren().remove(button);
gridPane.getChildren().removeAll();
```

- The program invokes the static `setHalignment` method to align the button right in cell (line 31).
- Example:

```
GridPane.setHalignment(button, HPos.RIGHT);
```

- Note that the scene size is not set (line 34). In this case, the scene size is automatically computed according to the sizes of the nodes placed inside the scene.

BorderPane

Using BorderPane in JavaFX

A `BorderPane` can place nodes in five regions: top, bottom, left, right, and center. Use the following methods to set nodes in these regions: `setTop(node)`, `setBottom(node)`, `setLeft(node)`, `setRight(node)`, `setCenter(node)`.

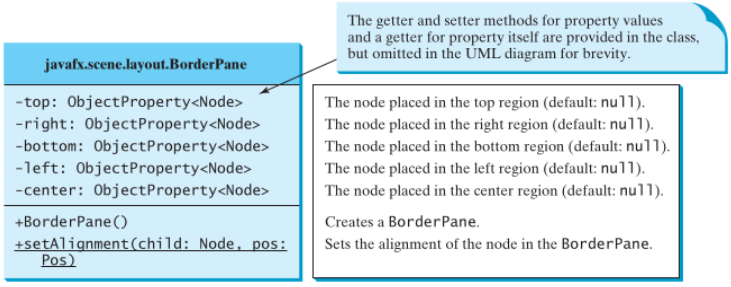


Figure 2: Class diagram for `BorderPane`

CustomPane and BorderPane in JavaFX

- The program defines `CustomPane` that extends `StackPane` (line 31).
- The constructor of `CustomPane`:
  - Adds a label with the specified title (line 33).
  - Sets a style for the border color.
  - Sets padding using insets (line 35).

```
class CustomPane extends StackPane {
    public CustomPane(String title) {
        getChildren().add(new Label(title));
        setStyle("-fx-border-color:
              black;");
        setPadding(new Insets(10, 10, 10,
              10));
    }
}
```

- The program creates a `BorderPane` (line 13) and places five instances of `CustomPane` into five regions of the border pane (lines 16–20).
- Example:

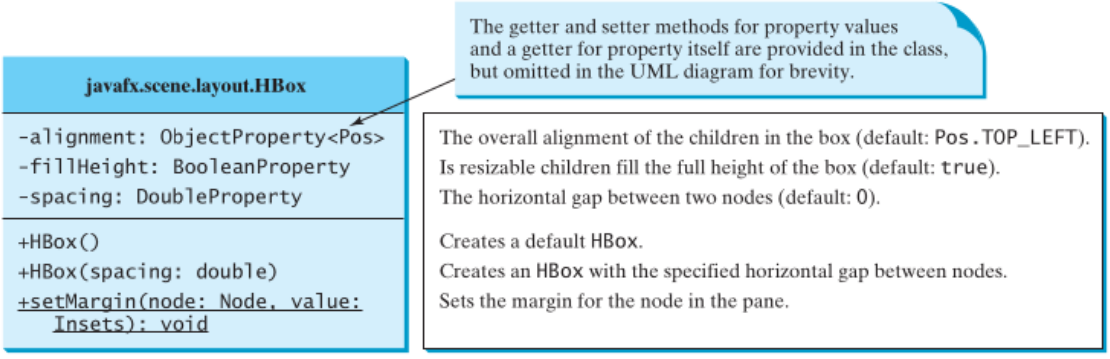
```
BorderPane borderPane = new BorderPane();
borderPane.setTop(new CustomPane("Top"));
borderPane.setBottom(new
      CustomPane("Bottom"));
borderPane.setLeft(new CustomPane("Left"));
borderPane.setRight(new
      CustomPane("Right"));
borderPane.setCenter(new
      CustomPane("Center"));
```

- A pane is a node, so a pane can be added into another pane.
- To remove a node from the top region, invoke `setTop(null)`.
- If a region is not occupied, no space will be allocated for this region.

HBox and VBox

HBox and VBox

An HBox lays out its children in a single horizontal row. A VBox lays out its children in a single vertical column. Recall that a FlowPane can lay out its children in multiple rows or multiple columns, but an HBox or a VBox can lay out children only in one row or one column. The class diagrams for HBox and VBox are shown in Figures 14.22 and 14.23.



Defining getHBox() and getVBox() Methods in JavaFX

getHBox() Method

- The `getHBox()` method returns an `HBox` that contains two buttons and an image view (lines 30–39).
- The background color of the `HBox` is set to gold using Java CSS (line 33).

```
public HBox getHBox() {
    HBox hBox = new HBox(15);
    hBox.setStyle("-fx-background-color:
        gold;");
    Button button1 = new Button("Button 1");
    Button button2 = new Button("Button 2");
    ImageView imageView = new ImageView(new
        Image("image/us.gif"));
    hBox.getChildren().addAll(button1,
        button2, imageView);
    return hBox;
}
```

getVBox() Method

- The `getVBox()` method returns a `VBox` that contains five labels (lines 41–55).
- The first label is added to the `VBox` in line 44 and the other four are added in line 51.
- The `setMargin` method is used to set a node's margin when placed inside the `VBox` (line 50).

```
public VBox getVBox() {
    VBox vBox = new VBox(15);
    Label label1 = new Label("Label 1");
    vBox.getChildren().add(label1);
    for (int i = 2; i <= 5; i++) {
        Label label = new Label("Label " +
            i);
        VBox.setMargin(label, new Insets(5,
            5, 5, 5));
        vBox.getChildren().add(label);
    }
    return vBox;
}
```

Shapes

Shape Class and Its Subclasses in JavaFX

- The `Shape` class is the abstract base class that defines the common properties for all shapes.
- Common properties include:
  - `fill` - Specifies a color that fills the interior of a shape.
  - `stroke` - Specifies a color that is used to outline a shape.
  - `strokeWidth` - Specifies the width of the outline of a shape.
- This section introduces the classes for drawing texts and simple shapes: `Text`, `Line`, `Rectangle`, `Circle`, `Ellipse`, `Arc`, `Polygon`, `Polyline`.

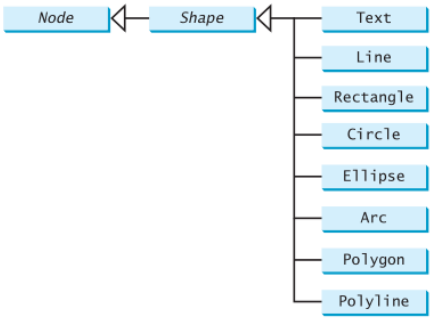


Figure 3: A shape is a node. The Shape class is the root of all shape classes.

The Text Class in JavaFX

- The `Text` class defines a node that displays a string at a starting point (x, y), as shown in Figure 14.27a.
- A `Text` object is usually placed in a pane.
- The pane's upper-left corner point is (0, 0) and the bottom-right point is (`pane.getWidth()`, `pane.getHeight()`).
- A string may be displayed in multiple lines separated by .
- The UML diagram for the `Text` class is shown in Figure 14.26.
- **ShowHBoxVBox.java** gives an example that demonstrates text.

```
Text text = new Text(50, 50, "Hello, JavaFX!");
Pane pane = new Pane();
pane.getChildren().add(text);
```

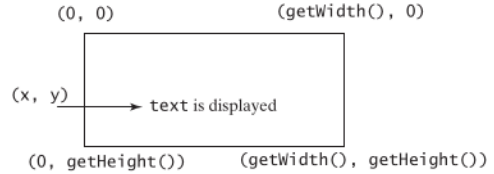


Figure 4: A Text object is created to display a text.

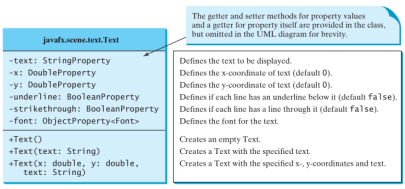


Figure 5: UML diagram for the `Text` class

The Text Class in JavaFX

- The `Text` class defines a node that displays a string at a starting point (x, y), as shown in Figure 14.27a.
- A `Text` object is usually placed in a pane.
- The pane's upper-left corner point is (0, 0), and the bottom-right point is (`pane.getWidth()`, `pane.getHeight()`).
- A string may be displayed in multiple lines separated by .

MC322 - Object Oriented Programming
Lesson 11.1
Graphical Interface - JavaFX Basics

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP