CCS3104
ADVANCED PROGRAMMING
Chapter 1 Basic GUI Programming

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

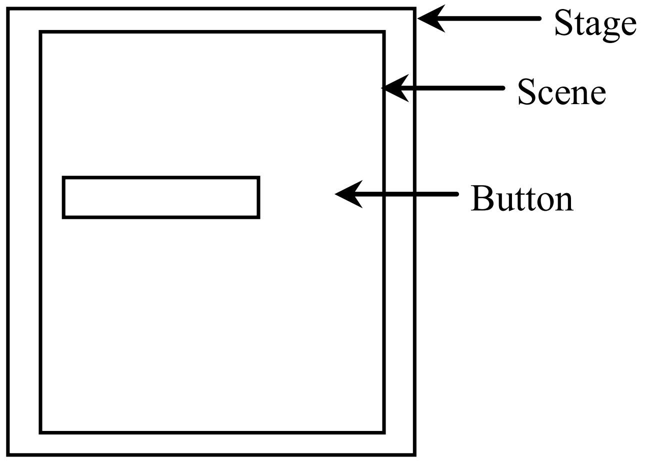
- To distinguish between JavaFX, Swing, and AWT (§14.2).
- To write a simple JavaFX program and understand the relationship among stages, scenes, and nodes (§14.3).
- To create user interfaces using panes, UI controls, and shapes (§14.4).
- To use the common properties **style** and **rotate** for nodes (§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).

JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications.
- When Java was introduced, the GUI classes were bundled in a library known as the *Abstract Windows Toolkit (AWT)*.
- AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
- AWT is prone to platform-specific bugs. The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*.
- Swing components are painted directly on canvases using Java code. Swing components depend less on the target platform and use less of the native GUI resource.
- With the release of Java 8, Swing is replaced by a completely new GUI platform known as *JavaFX*.

Basic Structure of JavaFX

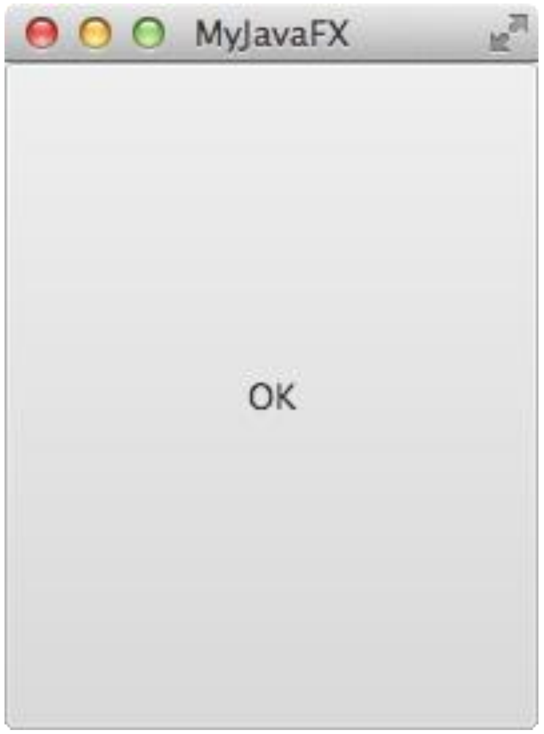
- Application
- Override the start(Stage) method
- Stage, Scene, and Nodes



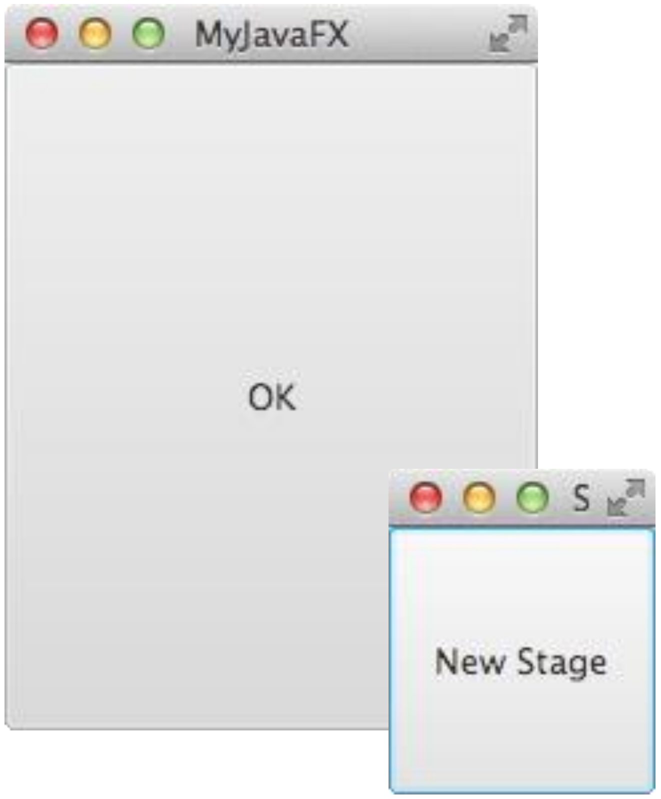
MyJavaFX Run

MultipleStageDemo Run

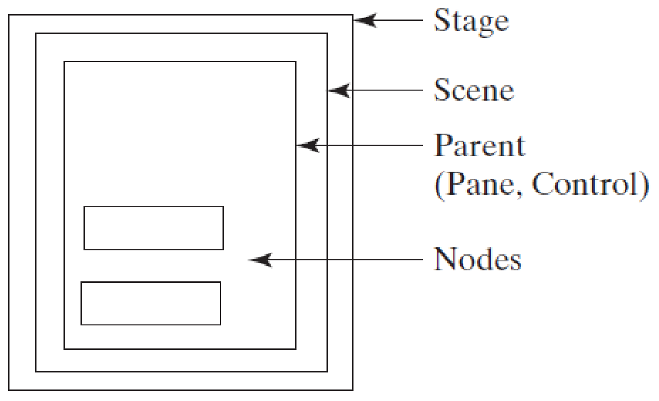
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MyJavaFX extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a button and place it in the scene
    Button btOK = new Button("OK");
    Scene scene = new Scene(btOK, 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



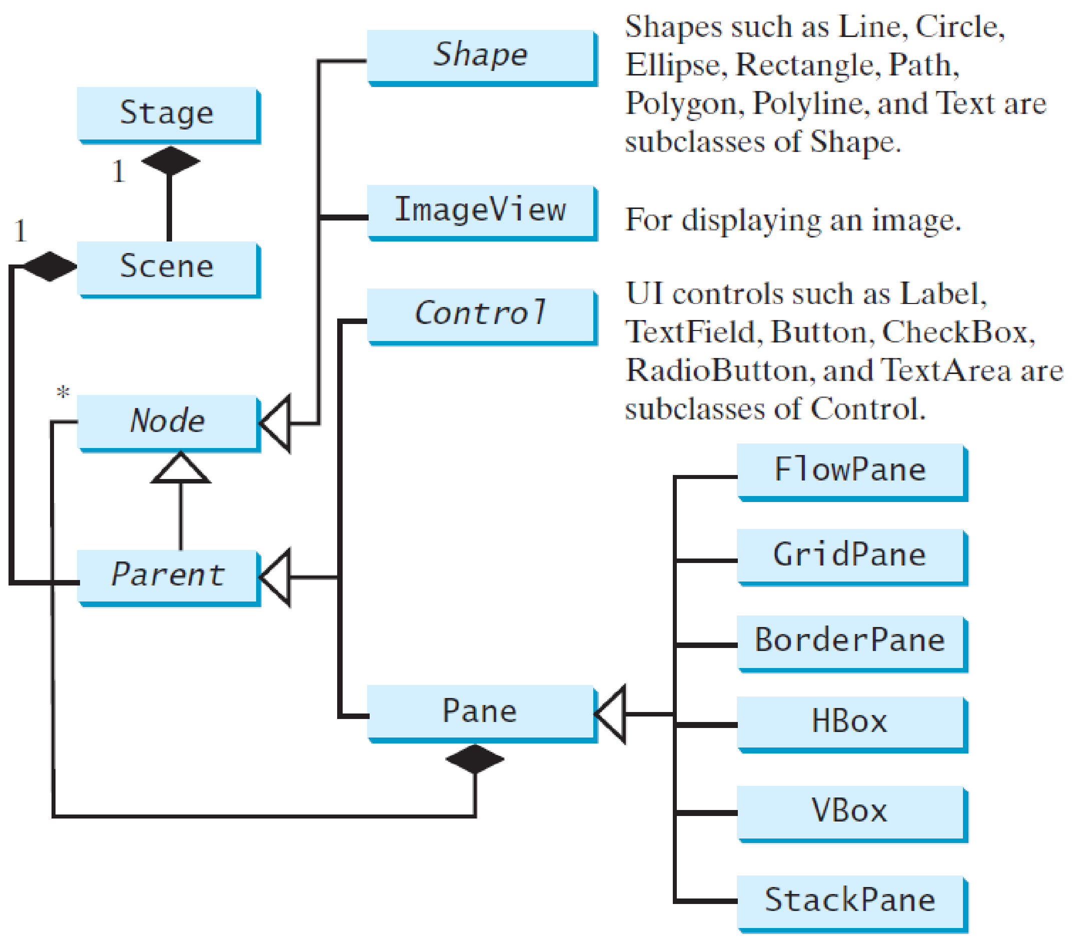
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MultipleStageDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place a button in the scene
    Scene scene = new Scene(new Button("OK"), 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

    Stage stage = new Stage(); // Create a new stage
    stage.setTitle("Second Stage"); // Set the stage title
    // Set a scene with a button in the stage
    stage.setScene(new Scene(new Button("New Stage"), 100, 100));
    stage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



Panes, UI Controls, and Shapes



(a)

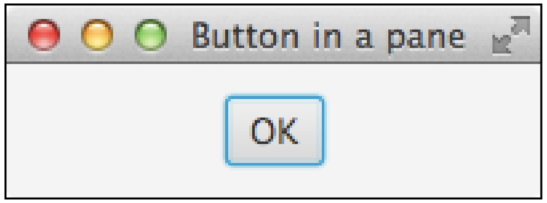

Stage
1
1
Scene
*
Node
Parent
Shape
ImageView
Control
Pane
FlowPane
GridPane
BorderPane
HBox
VBox
StackPane
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.
For displaying an image.
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

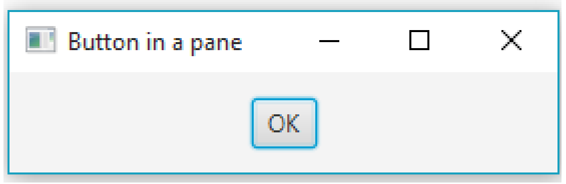
(b)

ButtonInPane

Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ButtonInPane extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place a button in the scene
    StackPane pane = new StackPane();
    pane.getChildren().add(new Button("OK"));
    Scene scene = new Scene(pane, 200, 50);
    primaryStage.setTitle("Button in a pane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



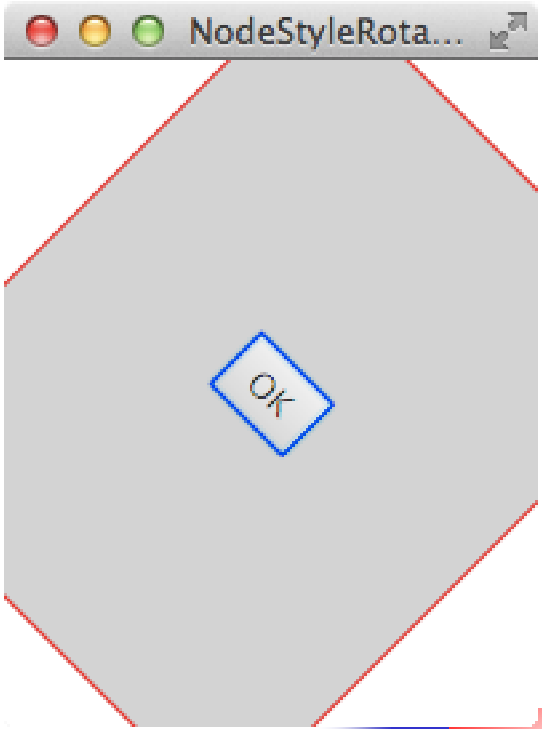


Common Properties and Methods for Nodes

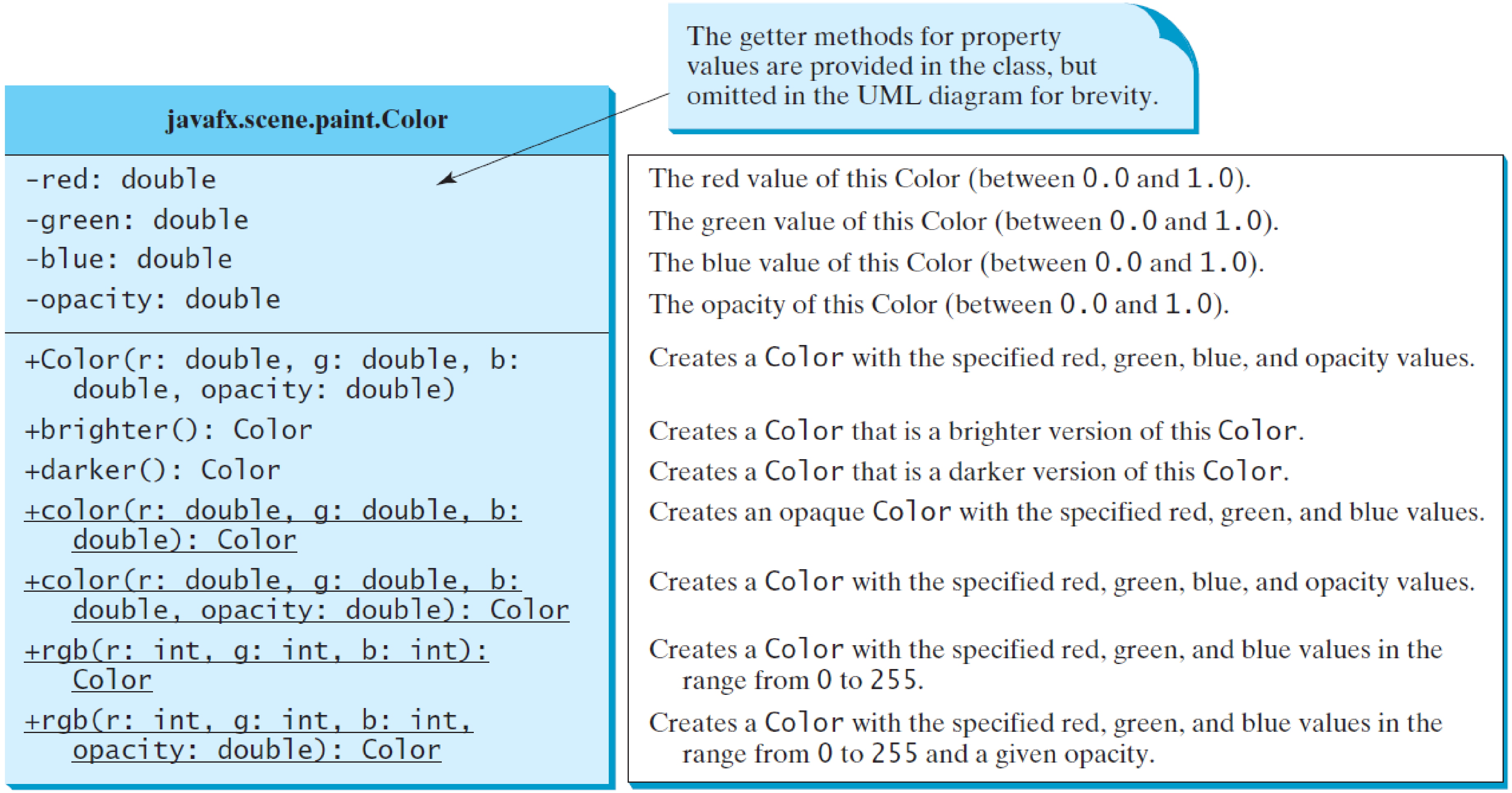
- style: set a JavaFX CSS style
- rotate: Rotate a node

NodeStyleRotateDemo Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;

public class NodeStyleRotateDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place a button in the scene
    StackPane pane = new StackPane();
    Button btOK = new Button("OK");
    btOK.setStyle("-fx-border-color: blue;");
    pane.getChildren().add(btOK);

    pane.setRotate(45);
    pane.setStyle(
      "-fx-border-color: red; -fx-background-color: lightgray;");

    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("NodeStyleRotateDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with
   * JavaFX support. Not needed for running from the
command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
```



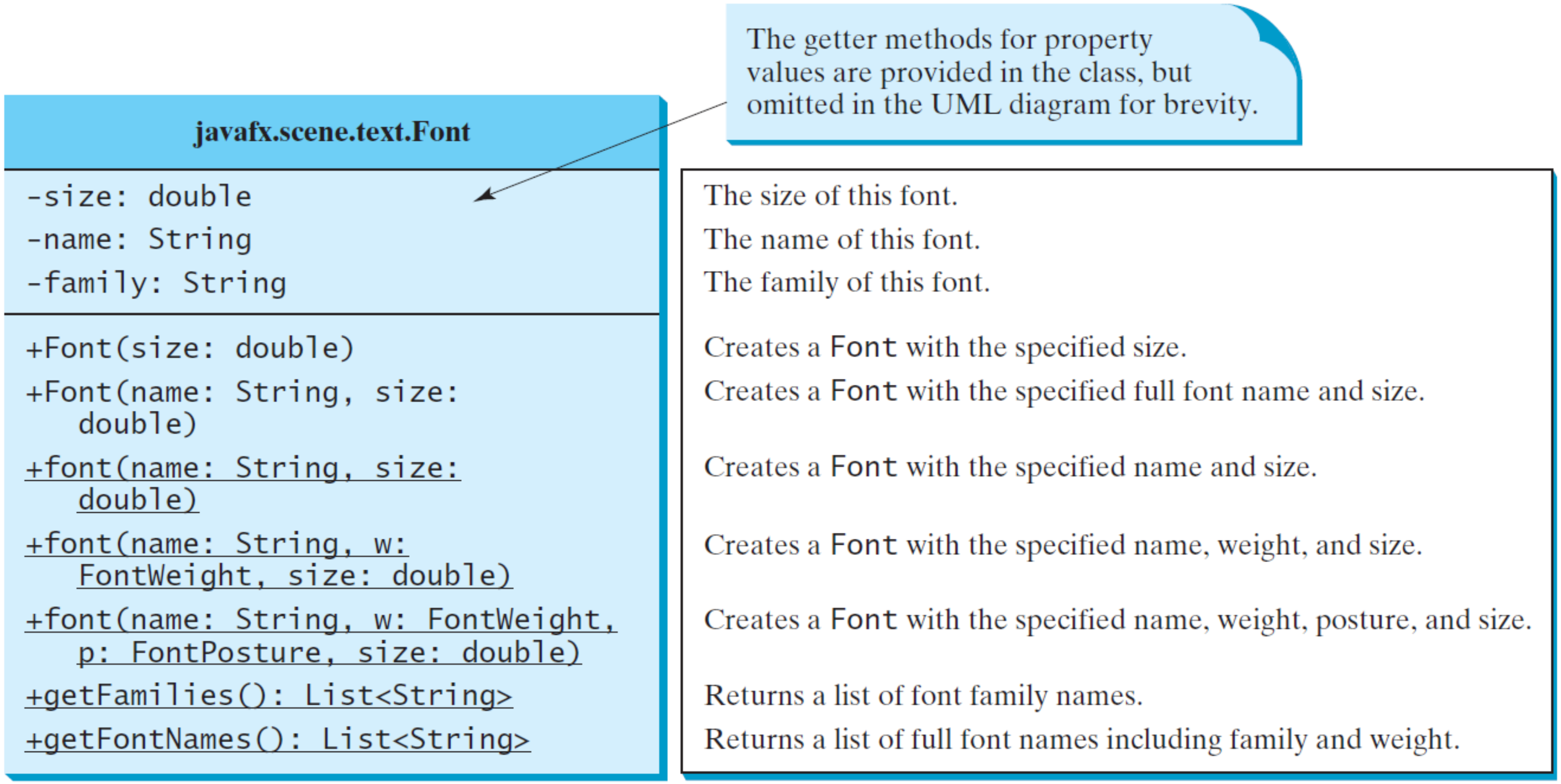
The Color Class



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

javafx.scene.paint.Color	
`-red: double`	The red value of this Color (between `0.0` and `1.0`).
`-green: double`	The green value of this Color (between `0.0` and `1.0`).
`-blue: double`	The blue value of this Color (between `0.0` and `1.0`).
`-opacity: double`	The opacity of this Color (between `0.0` and `1.0`).
`+Color(r: double, g: double, b: double, opacity: double)`	Creates a `Color` with the specified red, green, blue, and opacity values.
`+brighter(): Color`	Creates a `Color` that is a brighter version of this `Color`.
`+darker(): Color`	Creates a `Color` that is a darker version of this `Color`.
`+color(r: double, g: double, b: double): Color`	Creates an opaque `Color` with the specified red, green, and blue values.
`+color(r: double, g: double, b: double, opacity: double): Color`	Creates a `Color` with the specified red, green, blue, and opacity values.
`+rgb(r: int, g: int, b: int): Color`	Creates a `Color` with the specified red, green, and blue values in the range from `0` to `255`.
`+rgb(r: int, g: int, b: int, opacity: double): Color`	Creates a `Color` with the specified red, green, and blue values in the range from `0` to `255` and a given opacity.

The Font Class



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

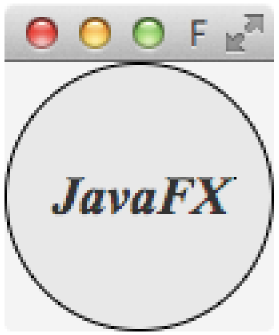
javafx.scene.text.Font	
`-size: double`	The size of this font.
`-name: String`	The name of this font.
`-family: String`	The family of this font.
`+Font(size: double)`	Creates a Font with the specified size.
`+Font(name: String, size: double)`	Creates a Font with the specified full font name and size.
`+font(name: String, size: double)`	Creates a Font with the specified name and size.
`+font(name: String, w: FontWeight, size: double)`	Creates a Font with the specified name, weight, and size.
`+font(name: String, w: FontWeight, p: FontPosture, size: double)`	Creates a Font with the specified name, weight, posture, and size.
`+getFamilies(): List<String>`	Returns a list of font family names.
`+getFontNames(): List<String>`	Returns a list of full font names including family and weight.

FontDemo

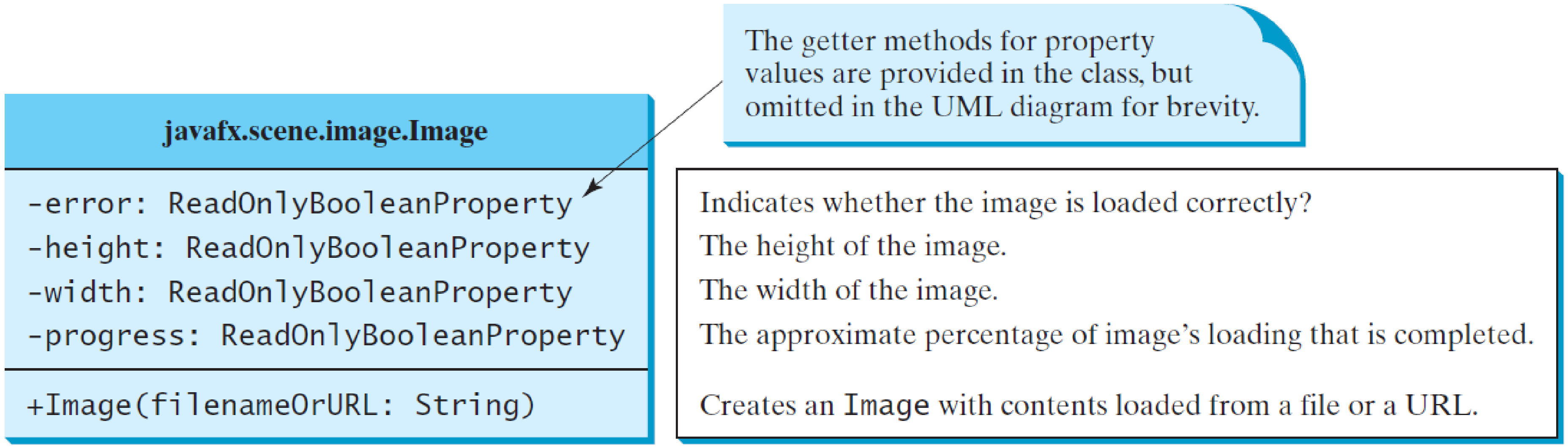
Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.*;
import javafx.scene.control.*;
import javafx.stage.Stage;

public class FontDemo extends Application {
  @Override
  // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new StackPane();

    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
    pane.getChildren().add(circle); // Add circle to the pane

    // Create a label and set its properties
    Label label = new Label("JavaFX");
    label.setFont(Font.font("Times New Roman",
      FontWeight.BOLD, FontPosture.ITALIC, 20));
    pane.getChildren().add(label);
```

```
// Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("FontDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



The Image Class

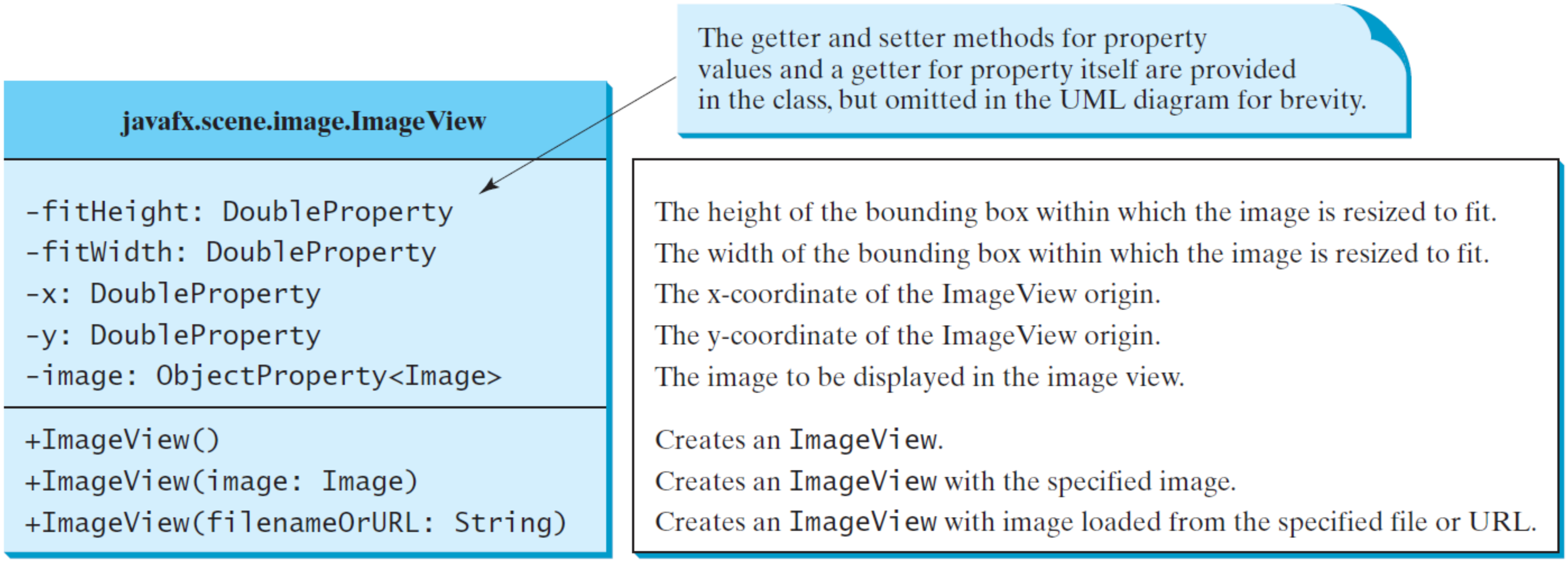


The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.image.Image	
`-error: ReadOnlyBooleanProperty`	Indicates whether the image is loaded correctly?
`-height: ReadOnlyBooleanProperty`	The height of the image.
`-width: ReadOnlyBooleanProperty`	The width of the image.
`-progress: ReadOnlyBooleanProperty`	The approximate percentage of image's loading that is completed.
`+Image(filenameOrURL: String)`	Creates an `Image` with contents loaded from a file or a URL.

Note: The images for this topic can be obtained from https://liveexample.pearsoncmg.com/common/image/, e.g. https://liveexample.pearsoncmg.com/common/image/us.gif.

The ImageView Class



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.

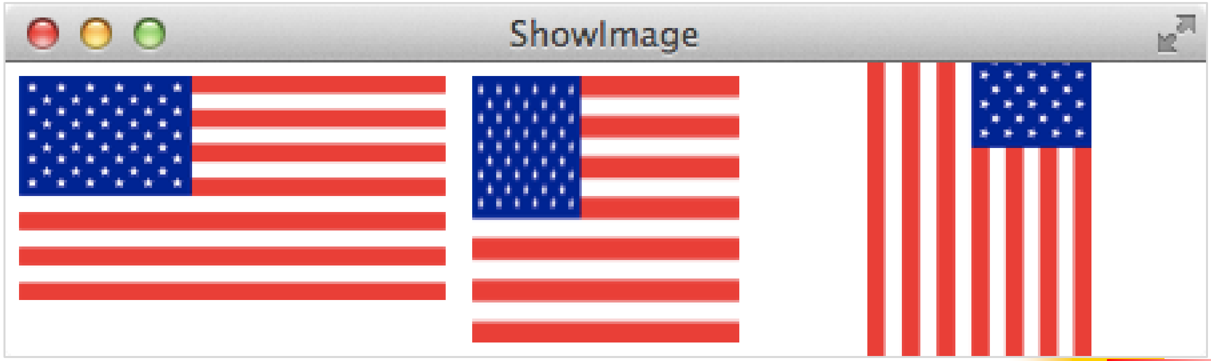
javafx.scene.image.ImageView

`-fitHeight: DoubleProperty`	The height of the bounding box within which the image is resized to fit.
`-fitWidth: DoubleProperty`	The width of the bounding box within which the image is resized to fit.
`-x: DoubleProperty`	The x-coordinate of the ImageView origin.
`-y: DoubleProperty`	The y-coordinate of the ImageView origin.
`-image: ObjectProperty<Image>`	The image to be displayed in the image view.
`+ImageView()`	Creates an `ImageView`.
`+ImageView(image: Image)`	Creates an `ImageView` with the specified image.
`+ImageView(filenameOrURL: String)`	Creates an `ImageView` with image loaded from the specified file or URL.

ShowImage

Run

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class ShowImage extends Application {
  @Override
  // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the image views
    Pane pane = new HBox(10);
    pane.setPadding(new Insets(5, 5, 5, 5));
    Image image = new Image("image/us.gif");
    pane.getChildren().add(new ImageView(image));

    ImageView imageView2 = new ImageView(image);
    imageView2.setFitHeight(100);
    imageView2.setFitWidth(100);
    pane.getChildren().add(imageView2);
    ImageView imageView3 = new ImageView(image);
    imageView3.setRotate(90);
    pane.getChildren().add(imageView3);

  // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowImage"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

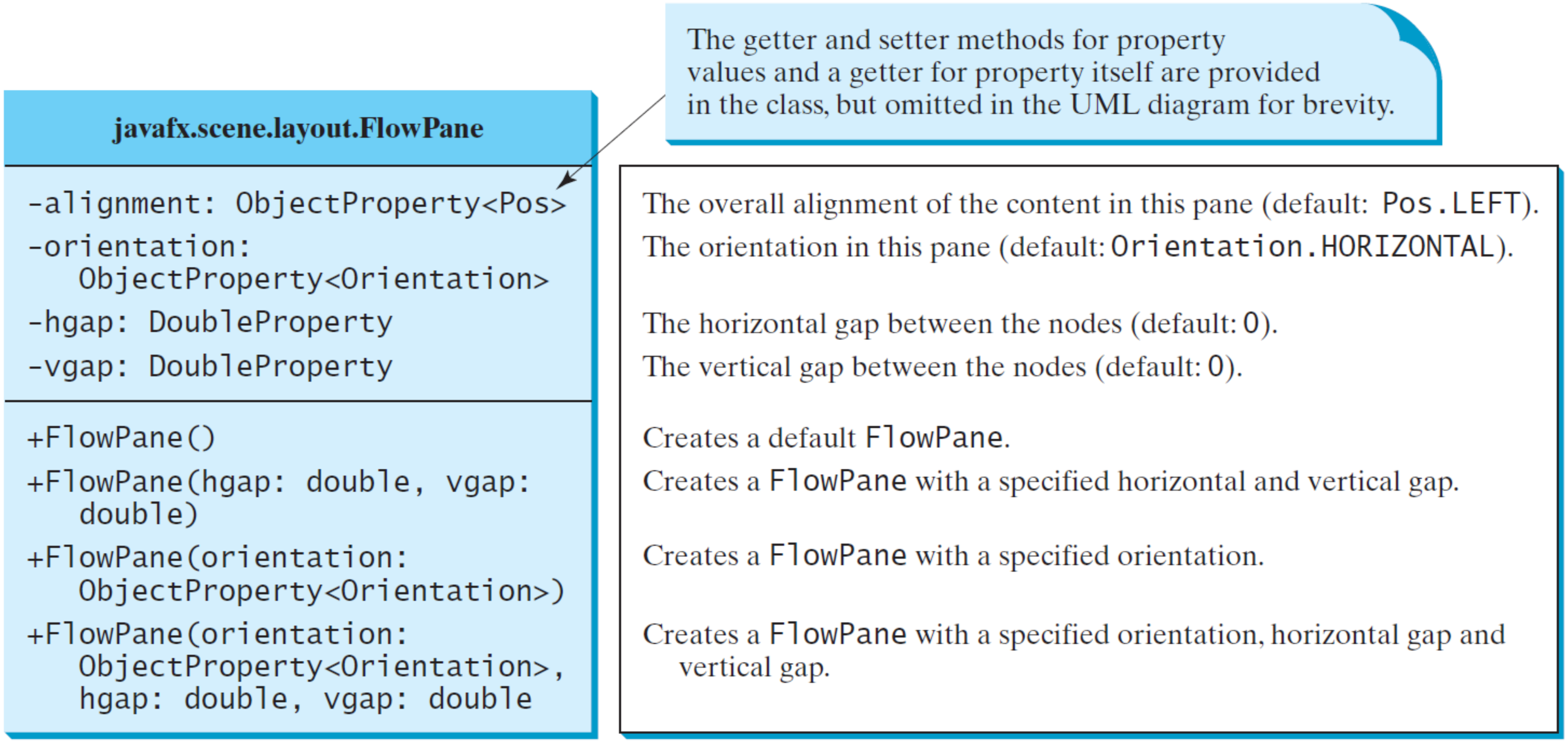


Layout Panes

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.

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.

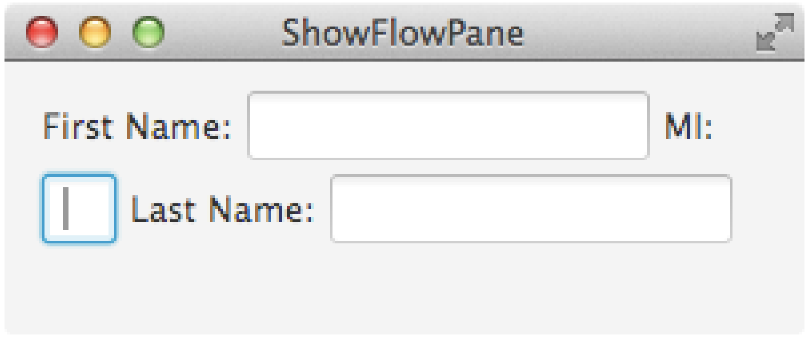
javafx.scene.layout.FlowPane

Member	Description
`-alignment: ObjectProperty<Pos>`	The overall alignment of the content in this pane (default: `Pos.LEFT`).
`-orientation: ObjectProperty<Orientation>`	The orientation in this pane (default: `Orientation.HORIZONTAL`).
`-hgap: DoubleProperty`	The horizontal gap between the nodes (default: `0`).
`-vgap: DoubleProperty`	The vertical gap between the nodes (default: `0`).
`+FlowPane()`	Creates a default `FlowPane`.
`+FlowPane(hgap: double, vgap: double)`	Creates a `FlowPane` with a specified horizontal and vertical gap.
`+FlowPane(orientation: ObjectProperty<Orientation>)`	Creates a `FlowPane` with a specified orientation.
`+FlowPane(orientation: ObjectProperty<Orientation>, hgap: double, vgap: double`	Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

ShowFlowPane

Run

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class ShowFlowPane extends Application {
  @Override
  // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);

    // Place nodes in the pane
    pane.getChildren().addAll(new Label("First Name:"),
      new TextField(), new Label("MI:"));
    TextField tfMi = new TextField();
    tfMi.setPrefColumnCount(1);
    pane.getChildren().addAll(tfMi, new Label("Last Name:"),
      new TextField());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("ShowFlowPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



GridPane

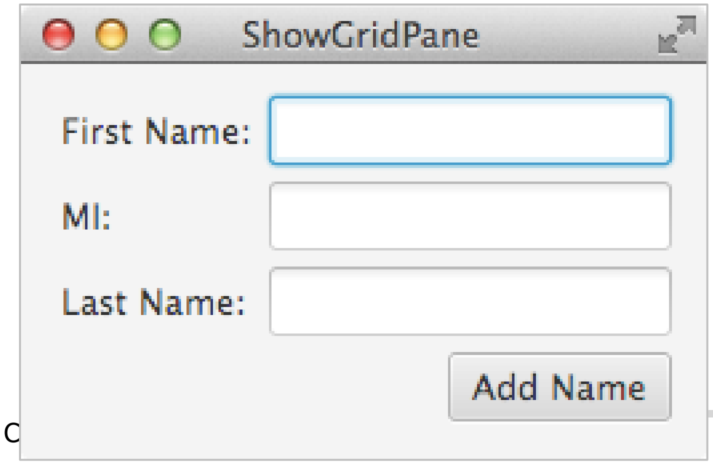
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane	
`-alignment: ObjectProperty<Pos>`	The overall alignment of the content in this pane (default: `Pos.LEFT`).
`-gridLinesVisible: BooleanProperty`	Is the grid line visible? (default: `false`)
`-hgap: DoubleProperty`	The horizontal gap between the nodes (default: 0).
`-vgap: DoubleProperty`	The vertical gap between the nodes (default: 0).
`+GridPane()`	Creates a `GridPane`.
`+add(child: Node, columnIndex: int, rowIndex: int): void`	Adds a node to the specified column and row.
`+addColumn(columnIndex: int, children: Node...): void`	Adds multiple nodes to the specified column.
`+addRow(rowIndex: int, children: Node...): void`	Adds multiple nodes to the specified row.
`+getColumnIndex(child: Node): int`	Returns the column index for the specified node.
`+setColumnIndex(child: Node, columnIndex: int): void`	Sets a node to a new column. This method repositions the node.
`+getRowIndex(child:Node): int`	Returns the row index for the specified node.
`+setRowIndex(child: Node, rowIndex: int): void`	Sets a node to a new row. This method repositions the node.
`+setHalighnment(child: Node, value: HPos): void`	Sets the horizontal alignment for the child in the cell.
`+setValighnment(child: Node, value: VPos): void`	Sets the vertical alignment for the child in the cell.

ShowGridPane

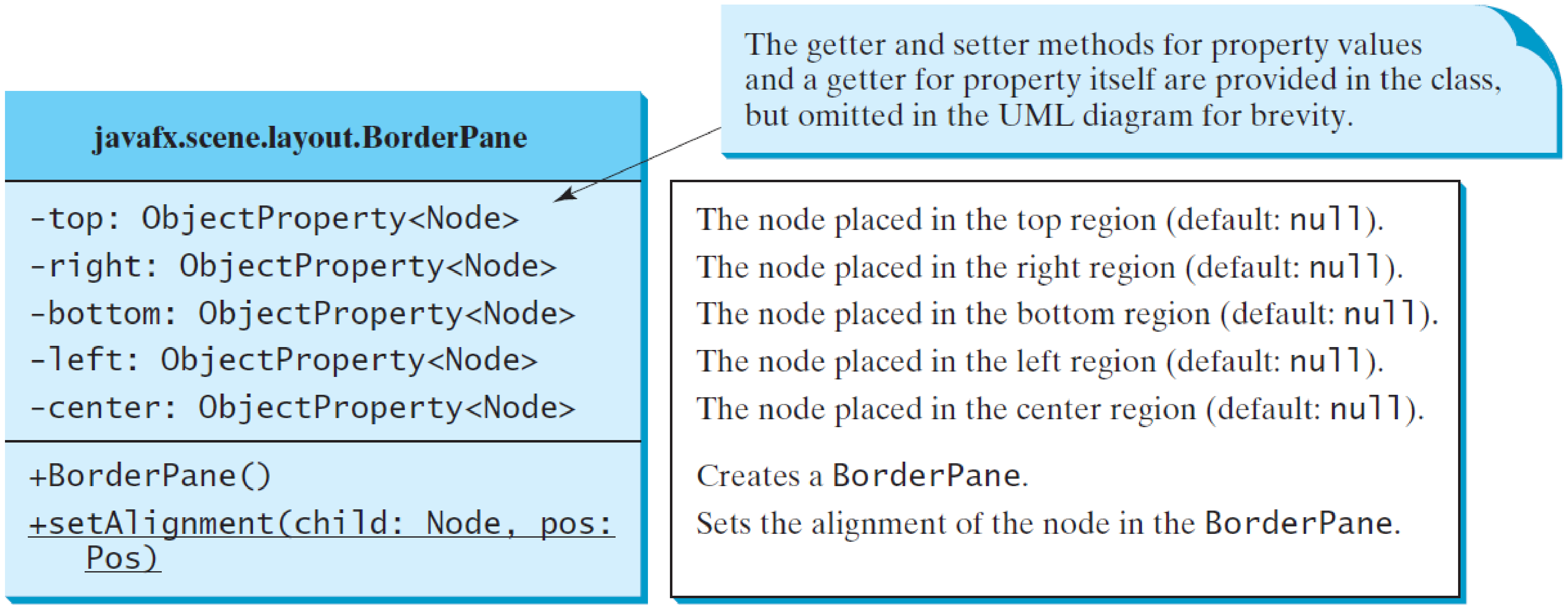
Run

```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ShowGridPane extends Application {
  @Override
  // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    GridPane pane = new GridPane();
    pane.setAlignment(Pos.CENTER);
    pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    pane.setHgap(5.5);
    pane.setVgap(5.5);
```



```
// Place nodes in the pane
    pane.add(new Label("First Name:"), 0, 0);
    pane.add(new TextField(), 1, 0);
    pane.add(new Label("MI:"), 0, 1);
    pane.add(new TextField(), 1, 1);
    pane.add(new Label("Last Name:"), 0, 2);
    pane.add(new TextField(), 1, 2);
    Button btAdd = new Button("Add Name");
    pane.add(btAdd, 1, 3);
    GridPane.setHalignment(btAdd, HPos.RIGHT);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowGridPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

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.

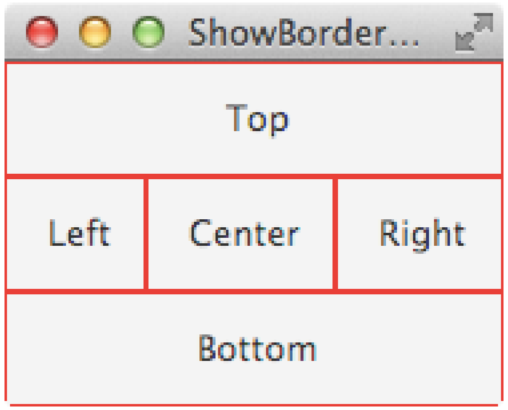
javafx.scene.layout.BorderPane	
`-top: ObjectProperty<Node>`	The node placed in the top region (default: `null`).
`-right: ObjectProperty<Node>`	The node placed in the right region (default: `null`).
`-bottom: ObjectProperty<Node>`	The node placed in the bottom region (default: `null`).
`-left: ObjectProperty<Node>`	The node placed in the left region (default: `null`).
`-center: ObjectProperty<Node>`	The node placed in the center region (default: `null`).
`+BorderPane()`	Creates a `BorderPane`.
`+setAlignment(child: Node, pos: Pos)`	Sets the alignment of the node in the `BorderPane`.

ShowBorderPane

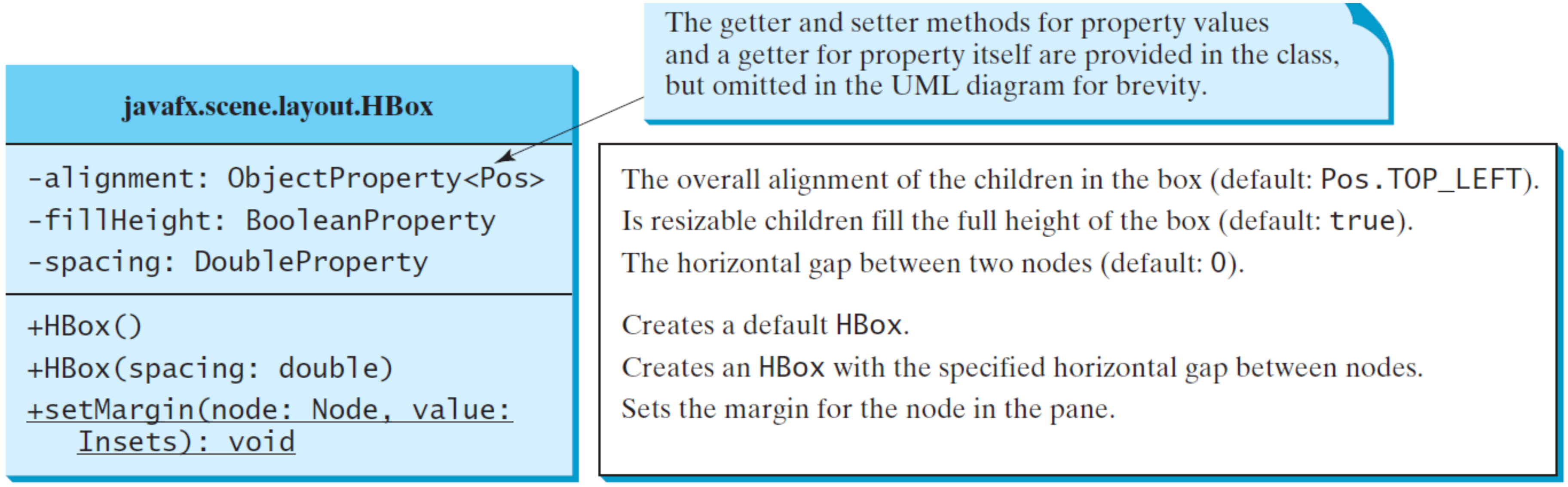
Run

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ShowBorderPane extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a border pane
    BorderPane pane = new BorderPane();

    // Place nodes in the pane
    pane.setTop(new CustomPane("Top"));
    pane.setRight(new CustomPane("Right"));
    pane.setBottom(new CustomPane("Bottom"));
    pane.setLeft(new CustomPane("Left"));
    pane.setCenter(new CustomPane("Center"));

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

```
// Define a custom pane to hold a label in the center of the
pane
class CustomPane extends StackPane {
  public CustomPane(String title) {
    getChildren().add(new Label(title));
    setStyle("-fx-border-color: red");
    setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
  }
}
```



HBox

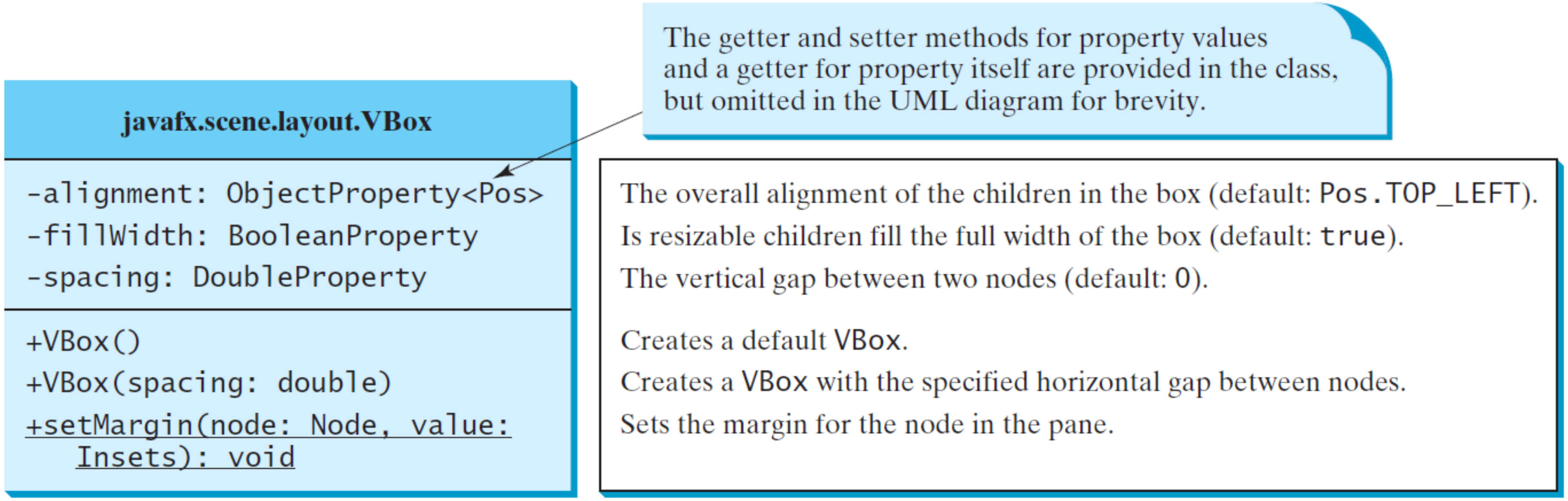


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.HBox

`-alignment: ObjectProperty<Pos>`	The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
`-fillHeight: BooleanProperty`	Is resizable children fill the full height of the box (default: `true`).
`-spacing: DoubleProperty`	The horizontal gap between two nodes (default: `0`).
`+HBox()`	Creates a default `HBox`.
`+HBox(spacing: double)`	Creates an `HBox` with the specified horizontal gap between nodes.
`+setMargin(node: Node, value: Insets): void`	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.

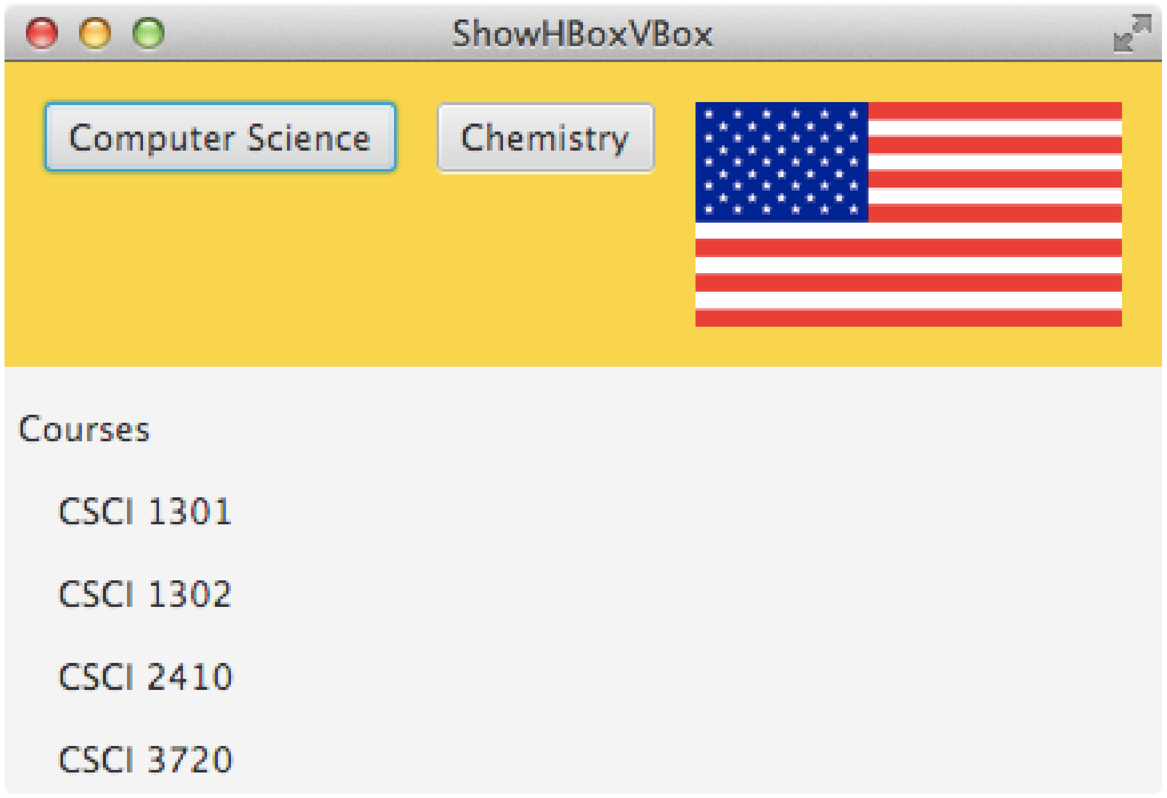
javafx.scene.layout.VBox	
-alignment: ObjectProperty<Pos>	The overall alignment of the children in the box (default: Pos.TOP_LEFT).
-fillWidth: BooleanProperty	Is resizable children fill the full width of the box (default: true).
-spacing: DoubleProperty	The vertical gap between two nodes (default: 0).
+VBox()	Creates a default VBox.
+VBox(spacing: double)	Creates a VBox with the specified horizontal gap between nodes.
<u>+setMargin(node: Node, value: Insets): void</u>	Sets the margin for the node in the pane.

ShowHBoxVBox

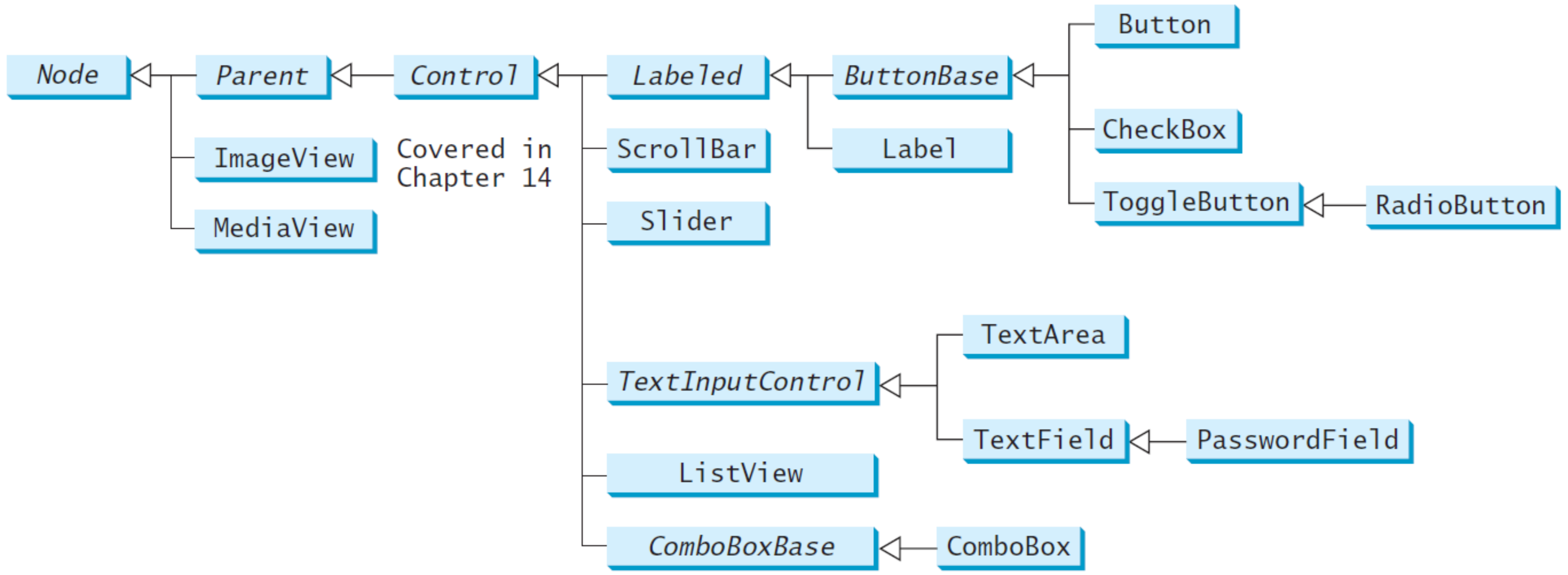
Run

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class ShowHBoxVBox extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a border pane
    BorderPane pane = new BorderPane();

    // Place nodes in the pane
    pane.setTop(getHBox());
    pane.setLeft(getVBox());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowHBoxVBox"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
```

```
 private HBox getHBox() {
   HBox hBox = new HBox(15);
   hBox.setPadding(new Insets(15, 15, 15, 15));
   hBox.setStyle("-fx-background-color: gold");
   hBox.getChildren().add(new Button("Computer Science"));
   hBox.getChildren().add(new Button("Chemistry"));
   ImageView imageView = new ImageView(new Image("image/us.gif"));
   hBox.getChildren().add(imageView);
   return hBox;
  }

  private VBox getVBox() {
   VBox vBox = new VBox(15);
   vBox.setPadding(new Insets(15, 5, 5, 5));
   vBox.getChildren().add(new Label("Courses"));

   Label[] courses = {new Label("CSCI 1301"), new Label("CSCI 1302"),
       new Label("CSCI 2410"), new Label("CSCI 3720")};

   for (Label course: courses) {
    VBox.setMargin(course, new Insets(0, 0, 0, 15));
    vBox.getChildren().add(course);
   }
   return vBox;
  }

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


ShowHBoxVBox
Computer Science
Chemistry
Courses
CSCI 1301
CSCI 1302
CSCI 2410
CSCI 3720

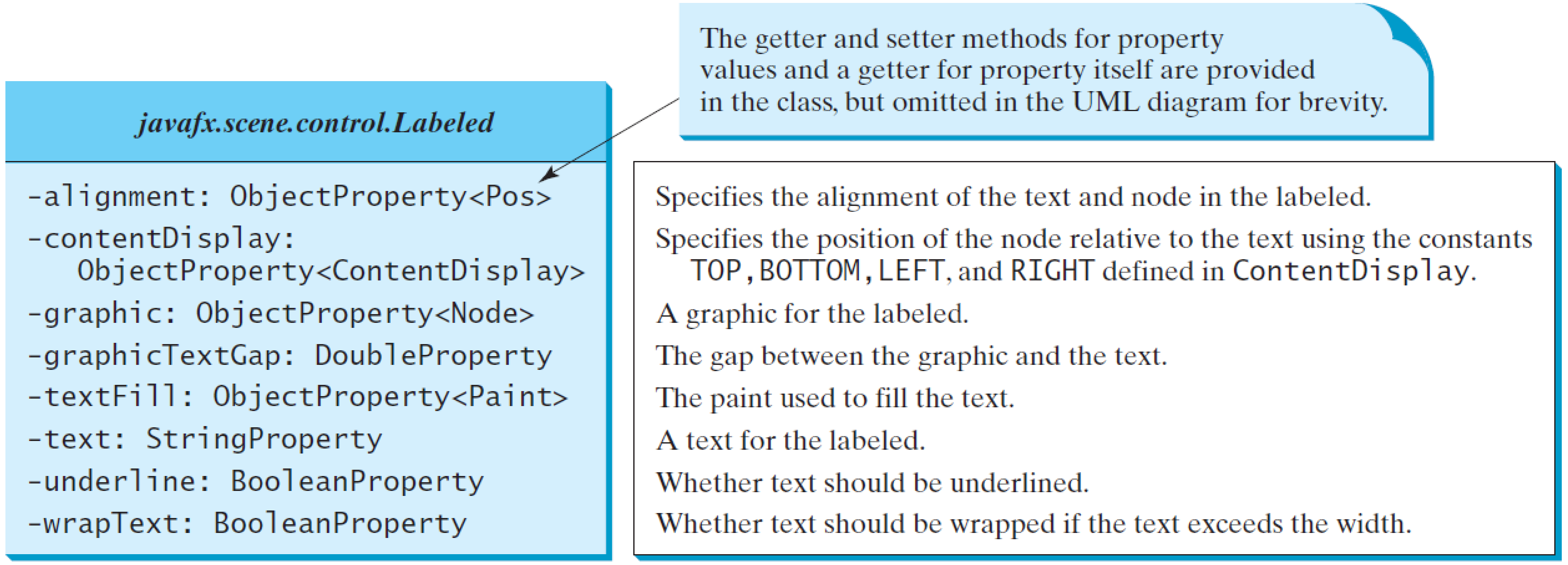
Frequently Used UI Controls



Throughout this book, the prefixes **lbl**, **bt**, **chk**, **rb**, **tf**, **pf**, **ta**, **cbo**, **lv**, **scb**, **sld**, and **mp** are used to name reference variables for **Label**, **Button**, **CheckBox**, **RadioButton**, **TextField**, **PasswordField**, **TextArea**, **ComboBox**, **ListView**, **ScrollBar**, **Slider**, and **MediaPlayer**.

Labeled

A *label* is a display area for a short text, a node, or both. It is often used to label other controls (usually text fields). Labels and buttons share many common properties. These common properties are defined in the **Labeled** class.



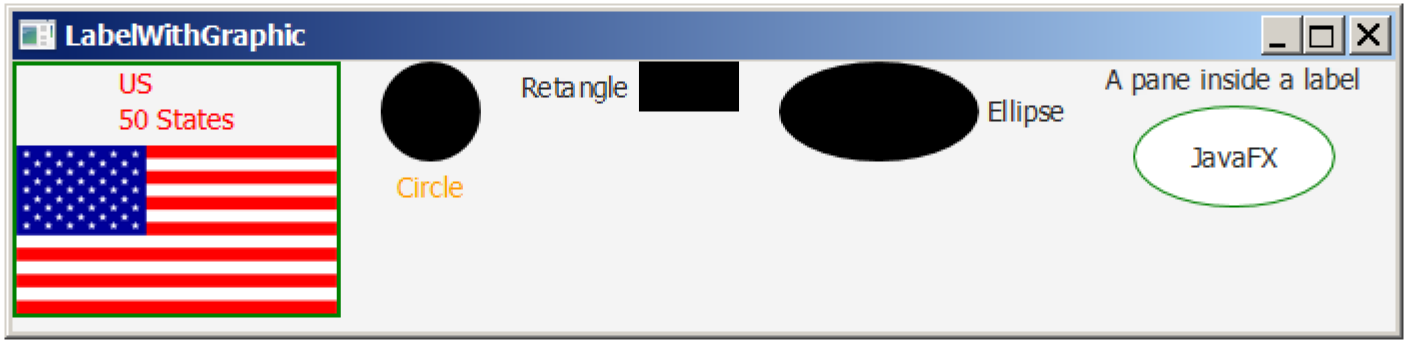
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

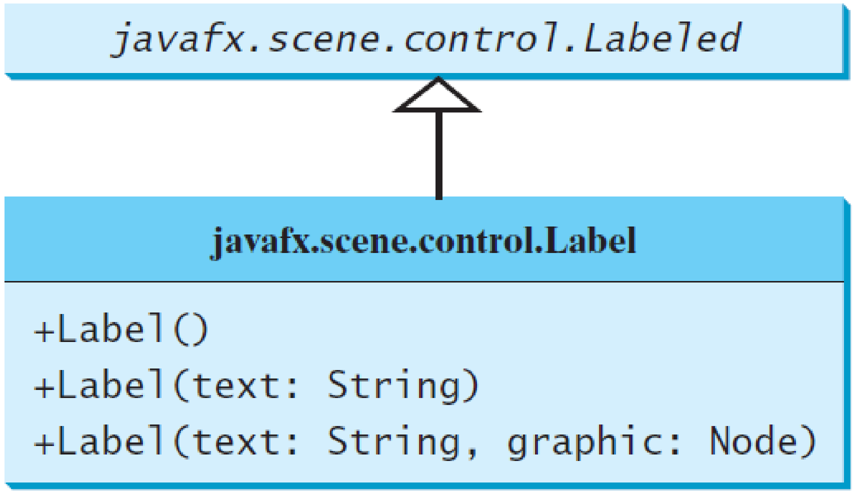
javafx.scene.control.Labeled

`-alignment: ObjectProperty<Pos>`	Specifies the alignment of the text and node in the labeled.
`-contentDisplay: ObjectProperty<ContentDisplay>`	Specifies the position of the node relative to the text using the constants `TOP`, `BOTTOM`, `LEFT`, and `RIGHT` defined in `ContentDisplay`.
`-graphic: ObjectProperty<Node>`	A graphic for the labeled.
`-graphicTextGap: DoubleProperty`	The gap between the graphic and the text.
`-textFill: ObjectProperty<Paint>`	The paint used to fill the text.
`-text: StringProperty`	A text for the labeled.
`-underline: BooleanProperty`	Whether text should be underlined.
`-wrapText: BooleanProperty`	Whether text should be wrapped if the text exceeds the width.

Label

The Label class defines labels.




javafx.scene.control.Labeled
javafx.scene.control.Label

+Label()	Creates an empty label.
+Label(text: String)	Creates a label with the specified text.
+Label(text: String, graphic: Node)	Creates a label with the specified text and graphic.

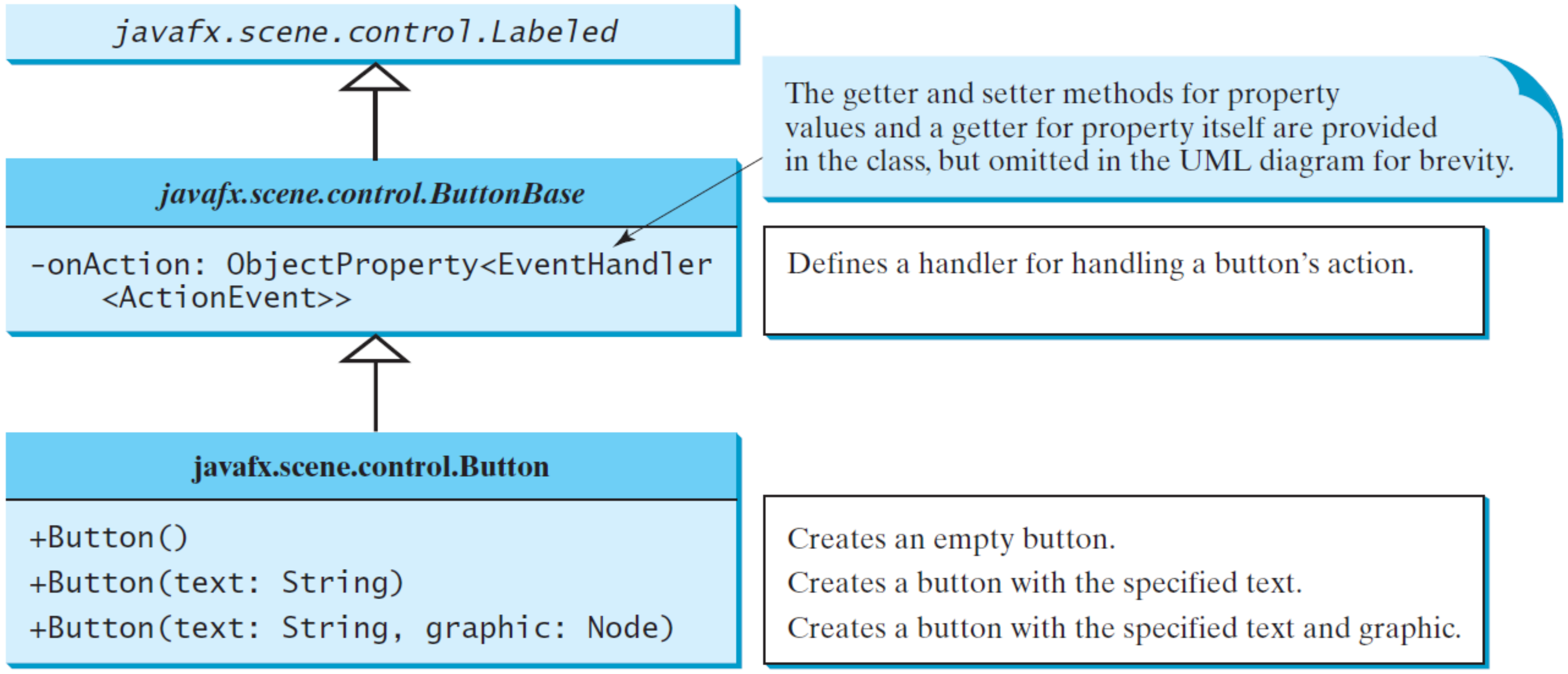
LabelWithGraphic Run

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Ellipse;

public class LabelWithGraphic extends Application {
 @Override // Override the start method in the Application class
 public void start(Stage primaryStage) {
  ImageView us = new ImageView(new Image("image/us.gif"));
  Label lb1 = new Label("US\n50 States", us);
  lb1.setStyle("-fx-border-color: green; -fx-border-width: 2");
  lb1.setContentDisplay(ContentDisplay.BOTTOM);
  lb1.setTextFill(Color.RED);

  Label lb2 = new Label("Circle", new Circle(50, 50, 25));
  lb2.setContentDisplay(ContentDisplay.TOP);
  lb2.setTextFill(Color.ORANGE);

  Label lb3 = new Label("Retangle", new Rectangle(10, 10, 50, 25));
  lb3.setContentDisplay(ContentDisplay.RIGHT);

  Label lb4 = new Label("Ellipse", new Ellipse(50, 50, 50, 25));
  lb4.setContentDisplay(ContentDisplay.LEFT);
```

```
Ellipse ellipse = new Ellipse(50, 50, 50, 25);
  ellipse.setStroke(Color.GREEN);
  ellipse.setFill(Color.WHITE);
  StackPane stackPane = new StackPane();
  stackPane.getChildren().addAll(ellipse, new Label("JavaFX"));
  Label lb5 = new Label("A pane inside a label", stackPane);
  lb5.setContentDisplay(ContentDisplay.BOTTOM);

  HBox pane = new HBox(20);
  pane.getChildren().addAll(lb1, lb2, lb3, lb4, lb5);

  // Create a scene and place it in the stage
  Scene scene = new Scene(pane, 450, 150);
  primaryStage.setTitle("LabelWithGraphic"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
 }

 /**
  * The main method is only needed for the IDE with limited
  * JavaFX support. Not needed for running from the command line.
  */
 public static void main(String[] args) {
  launch(args);
 }
}
```

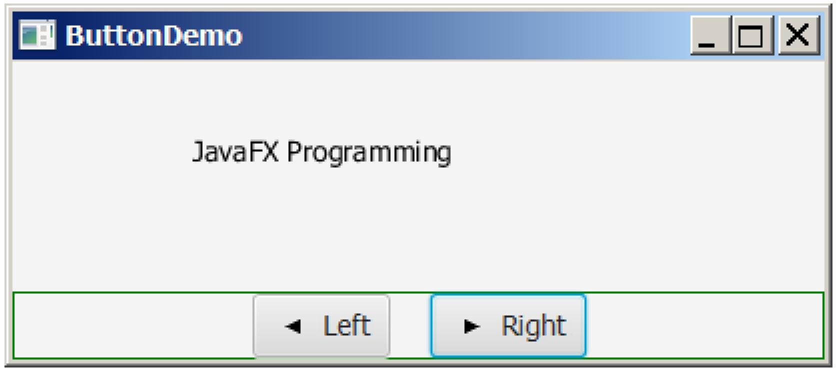
ButtonBase and Button

A *button* is a control that triggers an action event when clicked. JavaFX provides regular buttons, toggle buttons, check box buttons, and radio buttons. The common features of these buttons are defined in **ButtonBase** and **Labeled** classes.



javafx.scene.control.Labeled
javafx.scene.control.ButtonBase
-onAction: ObjectProperty<EventHandler <ActionEvent>>
javafx.scene.control.Button
+Button()
+Button(text: String)
+Button(text: String, graphic: Node)
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.
Defines a handler for handling a button's action.
Creates an empty button.
Creates a button with the specified text.
Creates a button with the specified text and graphic.

Button Example



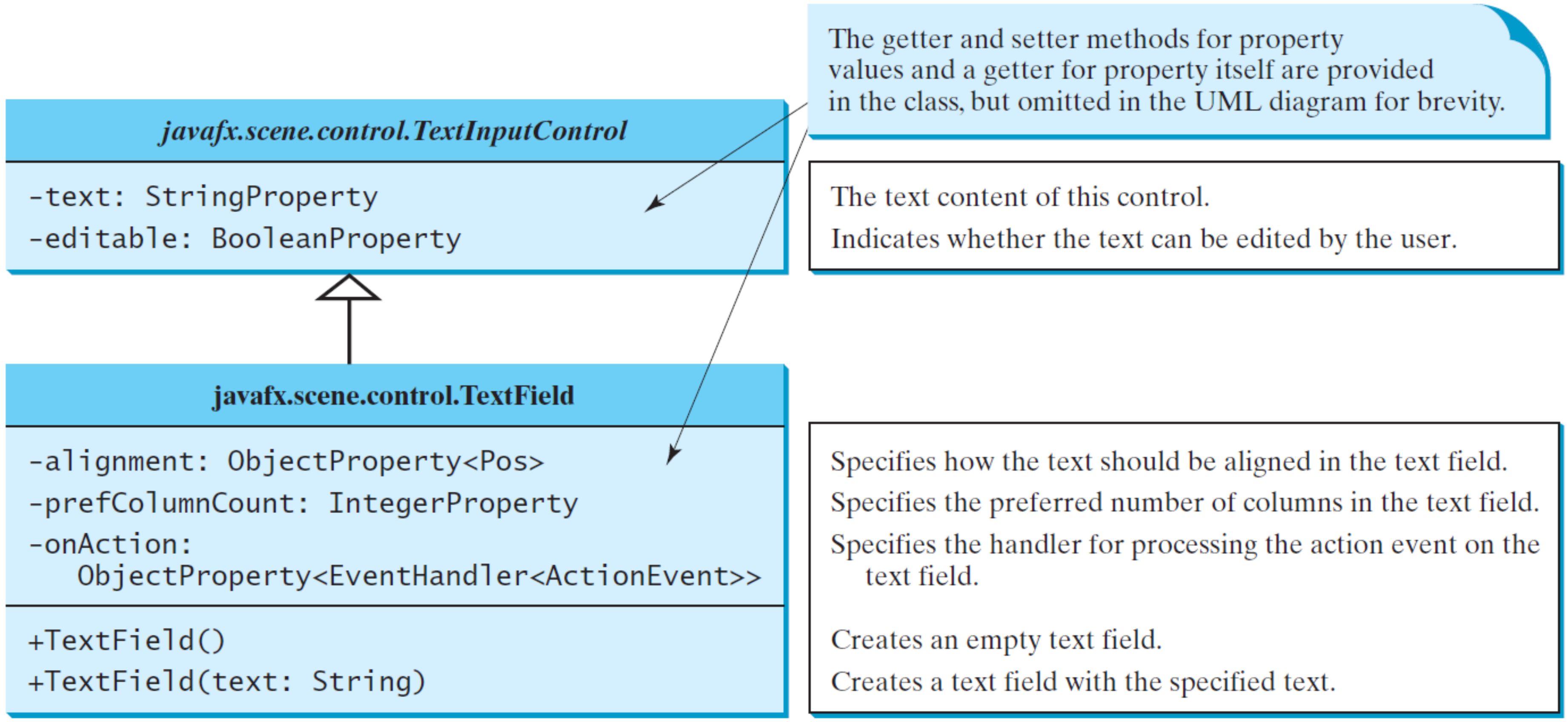
ButtonDemo Run

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;

public class ButtonDemo extends Application {
  protected Text text = new Text(50, 50, "JavaFX Programming");

  protected BorderPane getPane() {
    HBox paneForButtons = new HBox(20);
    Button btLeft = new Button("Left",
      new ImageView("image/left.gif"));
    Button btRight = new Button("Right",
      new ImageView("image/right.gif"));
    paneForButtons.getChildren().addAll(btLeft, btRight);
    paneForButtons.setAlignment(Pos.CENTER);
    paneForButtons.setStyle("-fx-border-color: green");

    BorderPane pane = new BorderPane();
    pane.setBottom(paneForButtons);

    Pane paneForText = new Pane();
    paneForText.getChildren().add(text);
    pane.setCenter(paneForText);

    btLeft.setOnAction(e -> text.setX(text.getX() - 10));
    btRight.setOnAction(e -> text.setX(text.getX() + 10));

    return pane;
  }
```

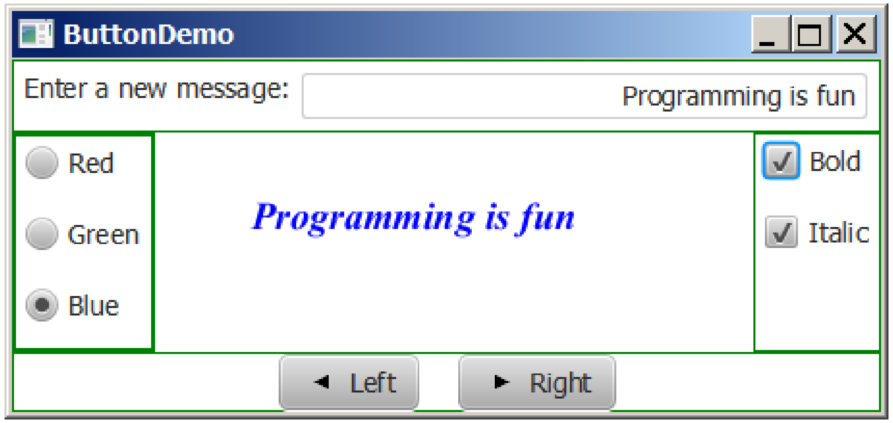
```java
@Override // Override the start method in the Application class
 public void start(Stage primaryStage) {
   // Create a scene and place it in the stage
   Scene scene = new Scene(getPane(), 450, 200);
   primaryStage.setTitle("ButtonDemo"); // Set the stage title
   primaryStage.setScene(scene); // Place the scene in the stage
   primaryStage.show(); // Display the stage
 }

 /**
  * The main method is only needed for the IDE with limited
  * JavaFX support. Not needed for running from the command line.
  */
 public static void main(String[] args) {
   launch(args);
 }
}
```

TextField

A text field can be used to enter or display a string. **TextField** is a subclass of **TextInputControl**.


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.control.TextInputControl
-text: StringProperty
-editable: BooleanProperty
The text content of this control.
Indicates whether the text can be edited by the user.
javafx.scene.control.TextField
-alignment: ObjectProperty<Pos>
-prefColumnCount: IntegerProperty
-onAction: ObjectProperty<EventHandler<ActionEvent>>
+TextField()
+TextField(text: String)
Specifies how the text should be aligned in the text field.
Specifies the preferred number of columns in the text field.
Specifies the handler for processing the action event on the text field.
Creates an empty text field.
Creates a text field with the specified text.

TextField Example



TextFieldDemo

Run

Summary

- Layout panes – FlowPane, GridPane, BorderPane, Hbox, Vbox and combinations of them
  - Esp. slides #20, #22, #24 and #27 – #28.
- Controls – Label, Button, TextField
  - Some related concepts will be covered in later chapters.

End of Chapter 1