Graphical User Interfaces JavaFX GUI Basics, Event Programming and GUI UI Controls

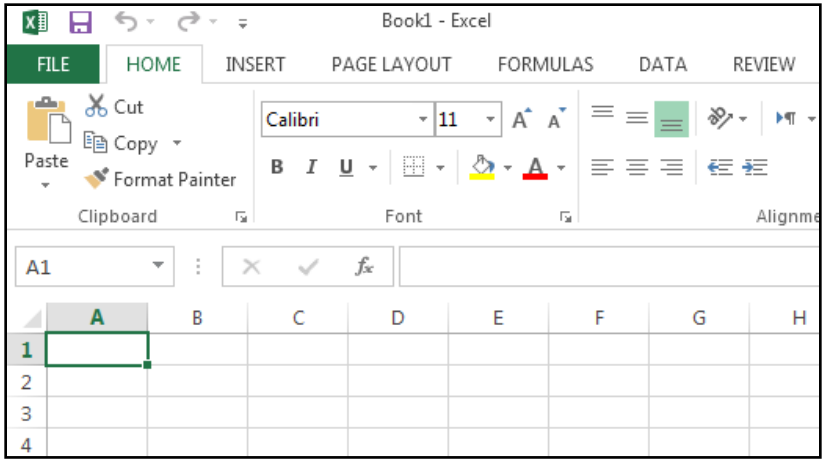
Paul Fodor

CSE260, Computer Science B: Honors

Stony Brook University

http://www.cs.stonybrook.edu/~cse260

GUI Examples

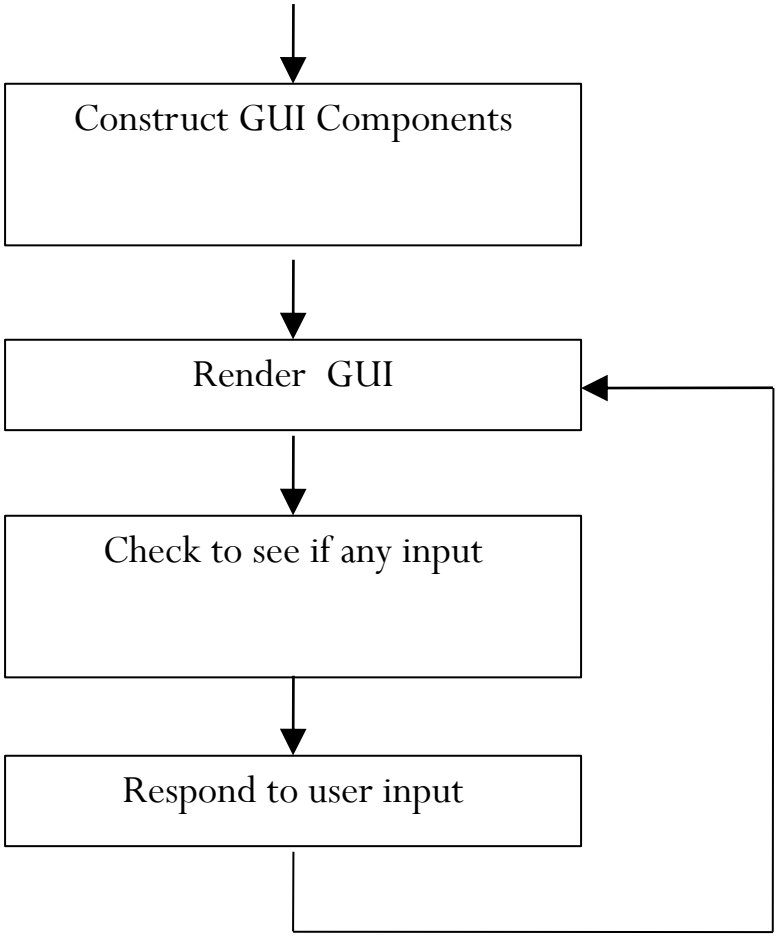


GUI

- Graphical User Interface (GUI)
  - provides user-friendly human interaction
- Building Java GUIs require use of frameworks:
  - AWT
  - Swing
  - JavaFX (part of Java since JSE 8, 2014) includes:
    - GUI components
    - Event Programming
    - Graphics

How do GUIs work?

- They loop and respond to events



Example: a mouse click on a button

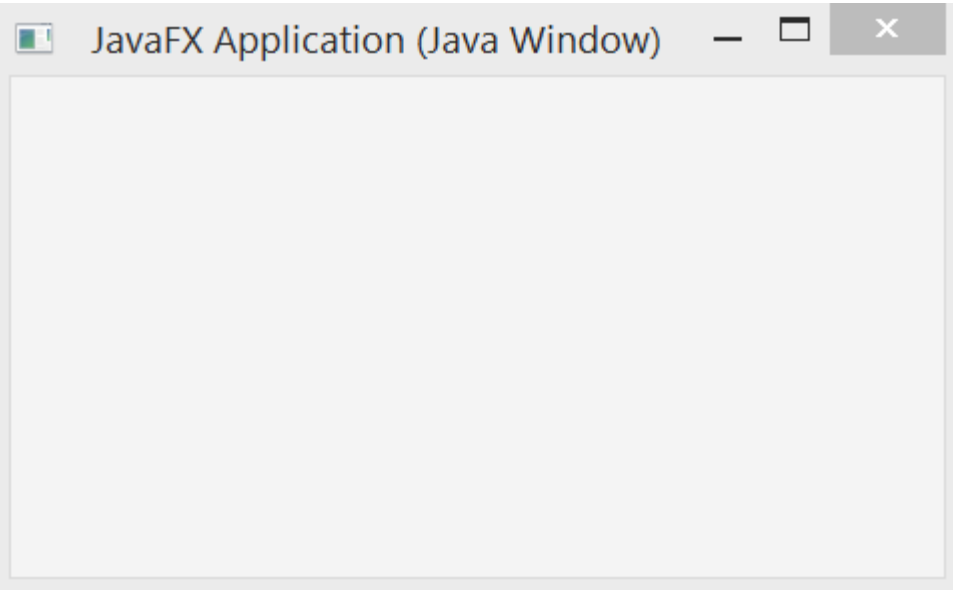
- Operating System recognizes mouse click
  - determines which window it was inside
  - notifies that program
- Program runs in loop
  - checks input buffer filled by OS
  - if it finds a mouse click:
    - determines which component in the program
    - if the click was on a relevant component
      - respond appropriately according to handler

GUI Look vs. Behavior

- Look
  - physical appearance
  - custom component design
  - containment
  - layout management
- Behavior
  - interactivity
  - event programmed response

What does a GUI framework do for you?

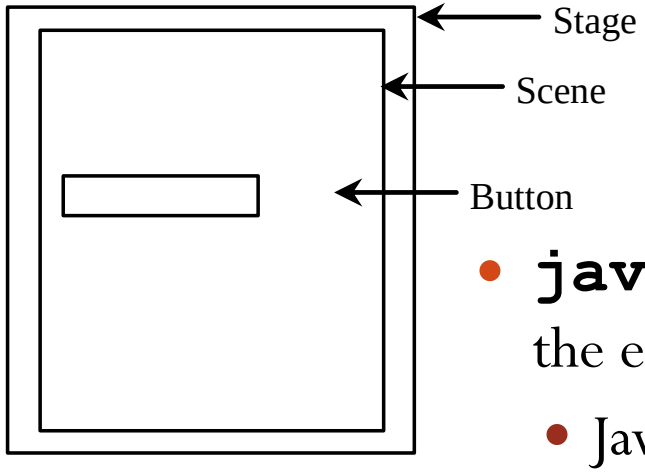
- Provides ready made visible, interactive, customizable components
  - you wouldn't want to have to code your own window



JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications in JDK8 (2014)
- History:
  - When Java was introduced (1996), the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT)
    - AWT was prone to platform-specific bugs
    - AWT was fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects
  - The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as Swing components (1997)
    - 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: JavaFX**

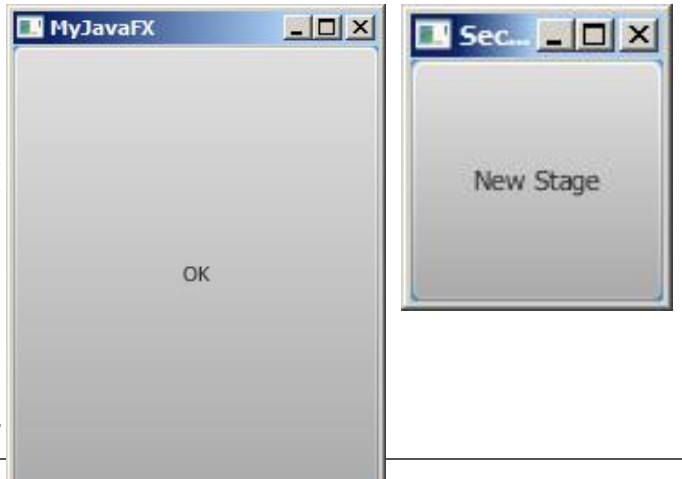
Basic Structure of JavaFX



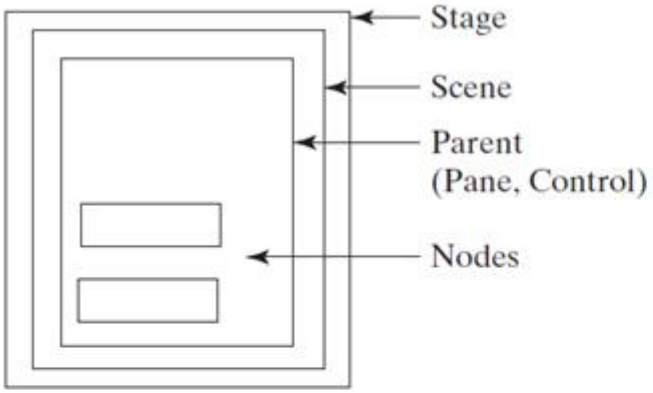
- **`javafx.application.Application`** is the entry point for JavaFX applications
  - JavaFX creates an application thread for running the application start method, processing input events, and running animation timelines.
  - Override the start(Stage) method!
- **`javafx.stage.Stage`** is the top level JavaFX container.
  - The primary Stage is constructed by the platform.
- **`javafx.scene.Scene`** class is the container for all content in a scene graph.
- **`javafx.scene.Node`** is the base class for scene graph nodes.

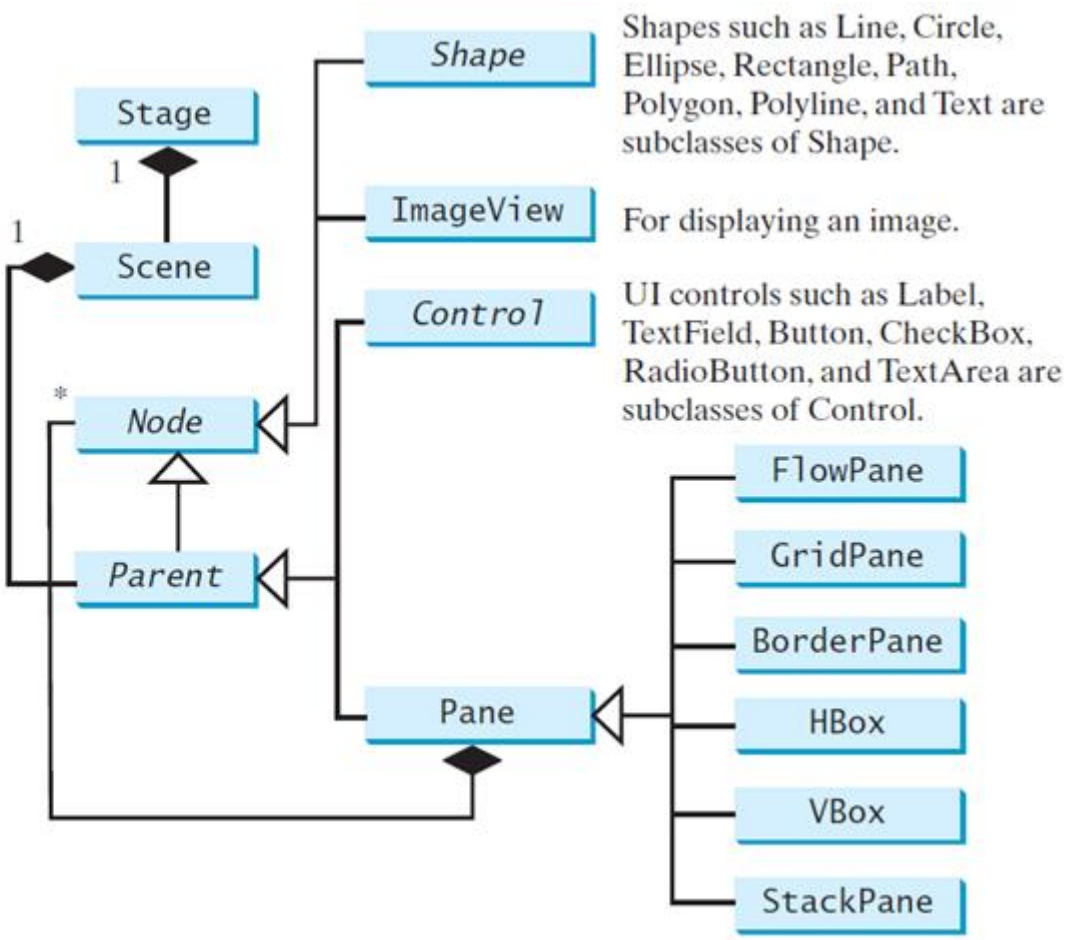
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

public class MyFirstJavaFX 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);
  }
}
```

```
// Multiple stages can be added beside the primaryStage
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

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
  }

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



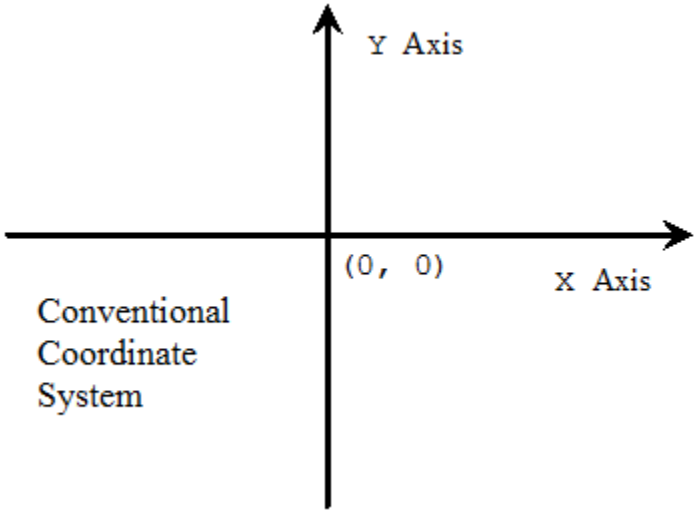
Panes, UI Controls, and Shapes



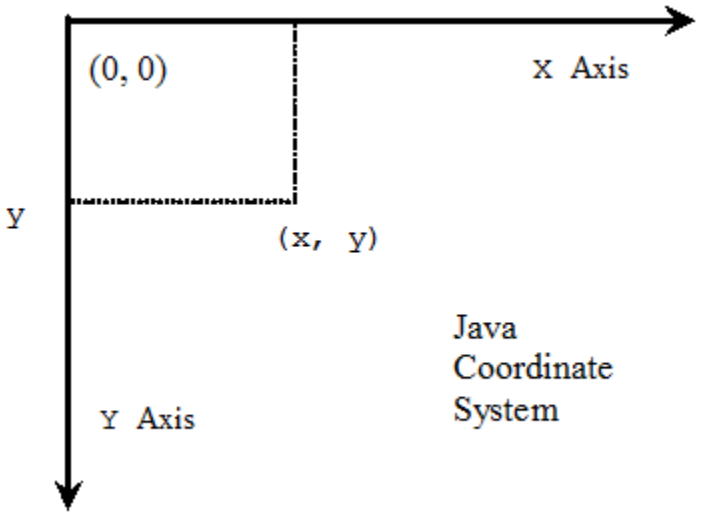


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

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
  }

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

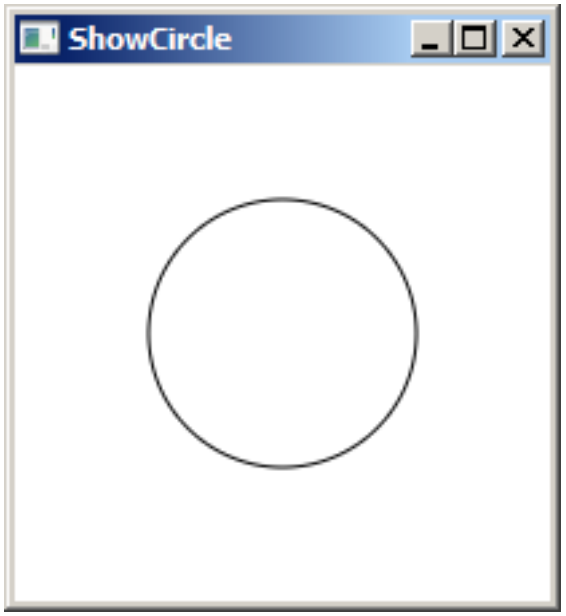
Display a Shape



- Programming Coordinate Systems start from the left-upper corner



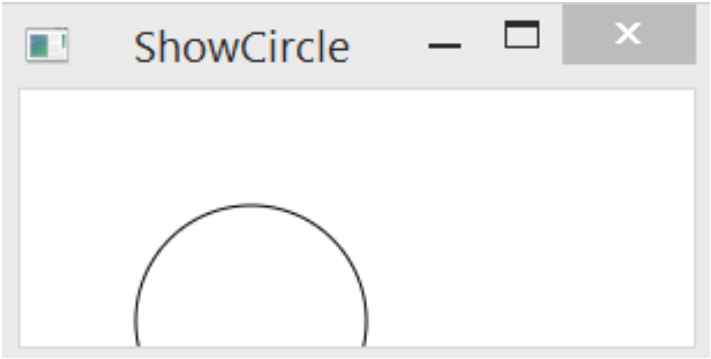
Circle in a Pane



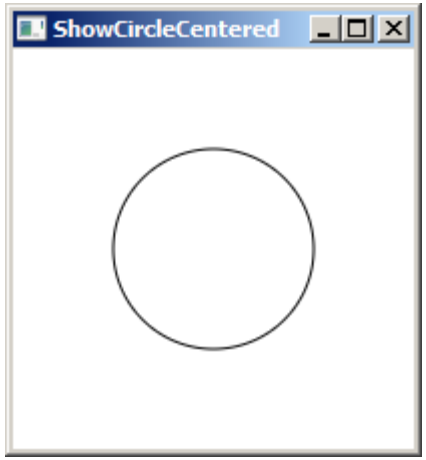

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;
import javafx.scene.paint.Color;

public class ShowCircle extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setCenterX(100);
    circle.setCenterY(100);
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(null);
    // Create a pane to hold the circle
    Pane pane = new Pane();
    pane.getChildren().add(circle);
    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircle"); // 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);
  }
}
```

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.
- Resizing the window in the previous example would cover the object:



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

public class ShowCircleCentered 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 Pane();
    // Create a circle and set its properties
    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"); // 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);
  }
}
```



JavaFX Beans and Binding

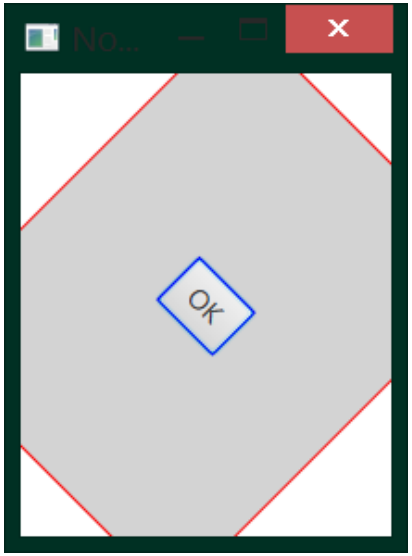
- Changes made to one object will automatically be reflected in another object
  - A graphical user interface automatically keeps its display synchronized with the application's underlying data: a binding observes its list of dependencies for changes, and then updates itself automatically after a change has been detected.

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BindingDemo {
  public static void main(String[] args) {
    DoubleProperty d1 = new SimpleDoubleProperty(1);
    DoubleProperty d2 = new SimpleDoubleProperty(2);
    d1.bind(d2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());

    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

Output:
d1 is 2.0 and d2 is 2.0

d1 is 70.2 and d2 is 70.2

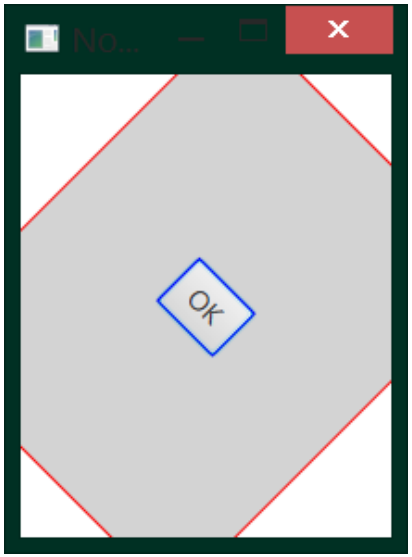
JavaFX CSS style and Node rotation




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
public class NodeStyleRotateDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    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 limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

JavaFX CSS style and Node rotation

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

public class NodeStyleRotateDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
```

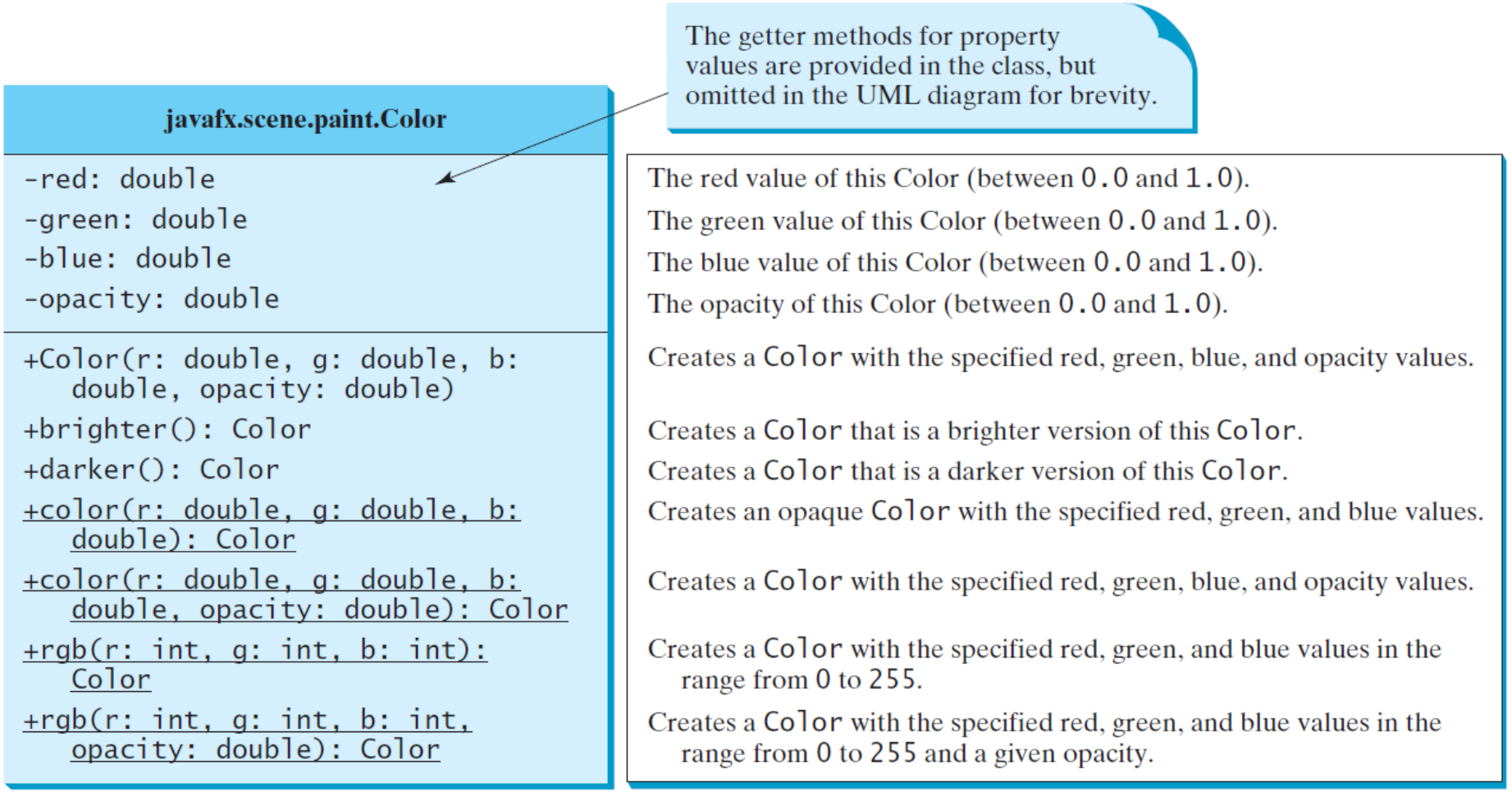



```
/* The StackPane layout pane places all of the nodes within
a single stack with each new node added on top of the
previous node. This layout model provides an easy way to
overlay text on a shape or image and to overlap common
shapes to create a complex shape. */
```

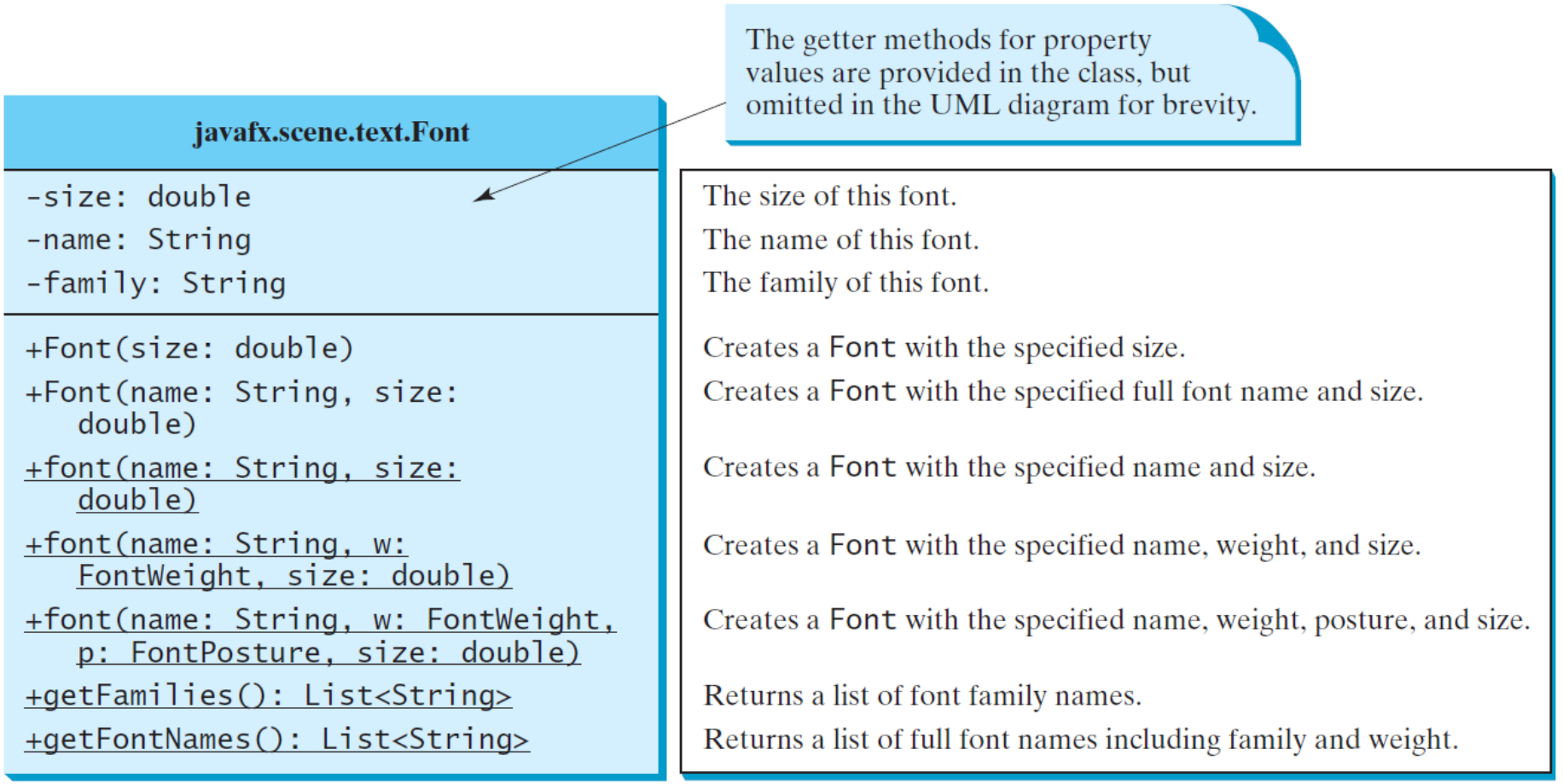
JavaFX External CSS style file

```
// Example to load and use a CSS style file in a scene
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;

public class ExternalCSSFile extends Application {
  @Override
  public void start(Stage primaryStage) {
    try {
      BorderPane root = new BorderPane();
      Scene scene = new Scene(root,400,400);
      scene.getStylesheets().add(getClass()
        .getResource("application.css").toExternalForm());
      primaryStage.setScene(scene);
      primaryStage.show();
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

Helper classes: The Color Class



Helper classes: The Font Class



The Image and ImageView Classes

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
-height: ReadOnlyBooleanProperty
-width: ReadOnlyBooleanProperty
-progress: ReadOnlyBooleanProperty
+Image(filenameOrURL: String)
```

Indicates whether the image is loaded correctly?
The height of the image.
The width of the image.
The approximate percentage of image's loading that is completed.

Creates an `Image` with contents loaded from a file or a URL.

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.

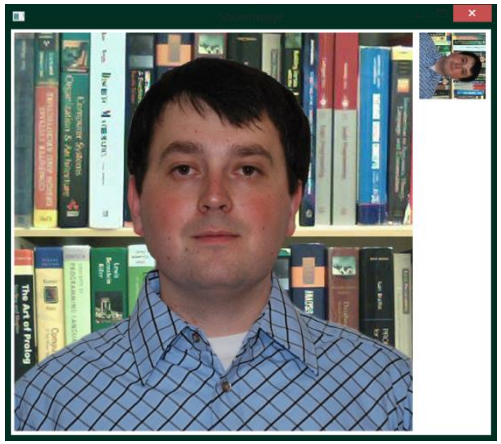
javafx.scene.image.ImageView

```
-fitHeight: DoubleProperty
-fitWidth: DoubleProperty
-x: DoubleProperty
-y: DoubleProperty
-image: ObjectProperty<Image>
+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
```

The height of the bounding box within which the image is resized to fit.
The width of the bounding box within which the image is resized to fit.
The x-coordinate of the ImageView origin.
The y-coordinate of the ImageView origin.
The image to be displayed in the image view.

Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.

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

public class ShowImage extends Application {
  @Override
  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("paul.jpg");
    pane.getChildren().add(new ImageView(image));
    ImageView imageView2 = new ImageView(image);
    imageView2.setFitHeight(100);
    imageView2.setFitWidth(100);
    imageView2.setRotate(90);
    pane.getChildren().add(imageView2);
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowImage");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

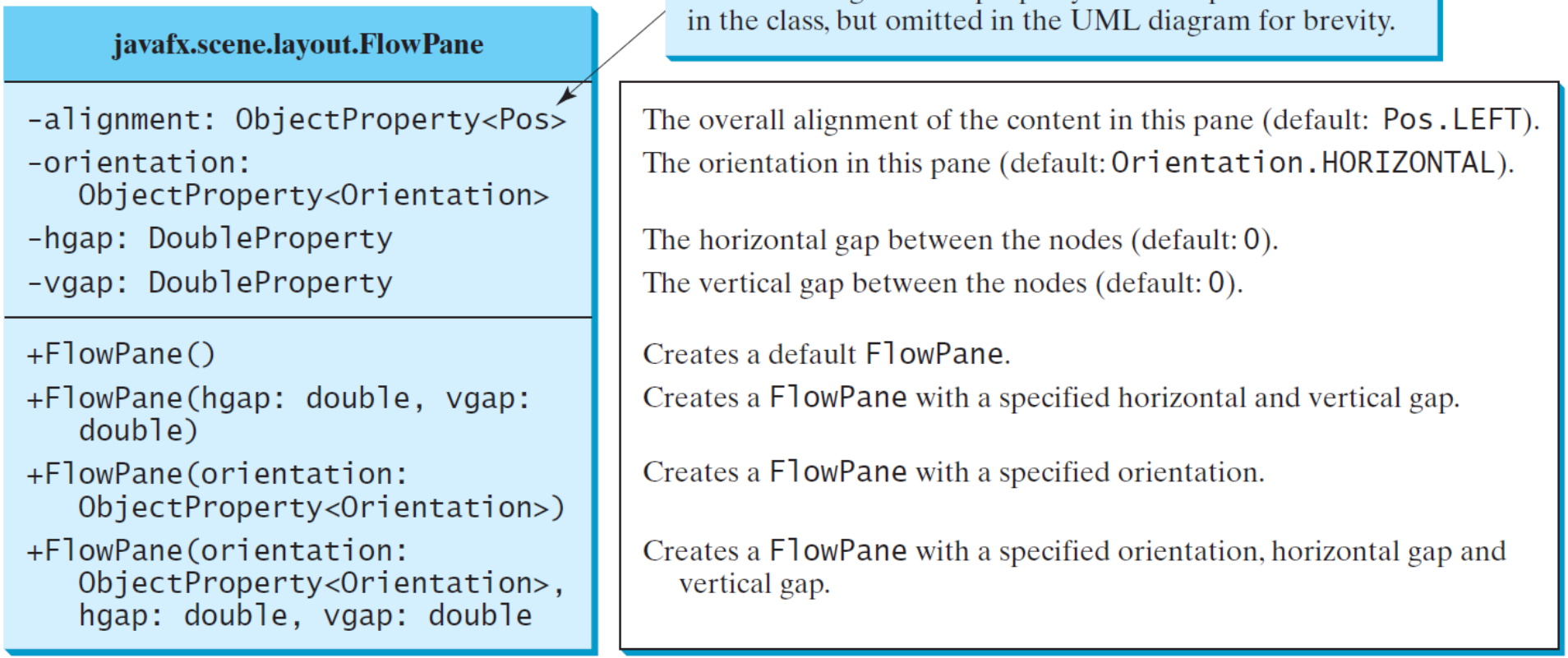


Layout Panes

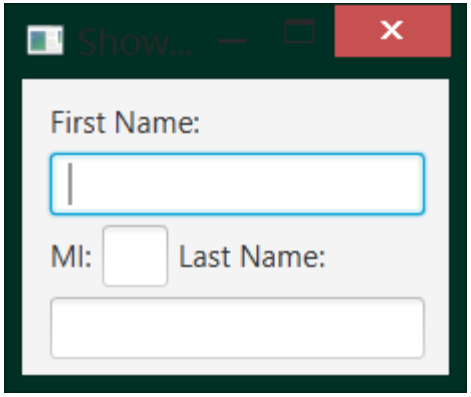
- JavaFX provides many **types of panes for organizing nodes in a container.**

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



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Insets;
public class ShowFlowPane extends Application {
  @Override
  public void start(Stage primaryStage) {
    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, 210, 150);
    primaryStage.setTitle("ShowFlowPane");
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
    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>
-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.

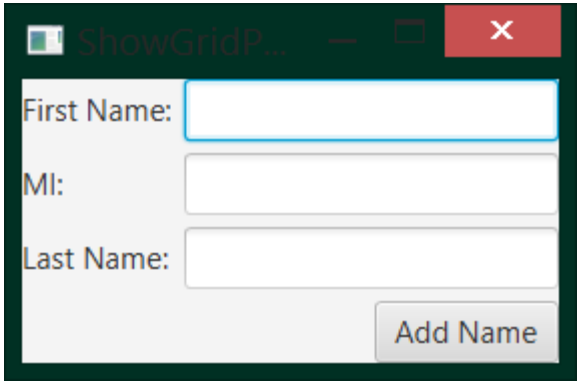
Sets a node to a new column. This method repositions the node.

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

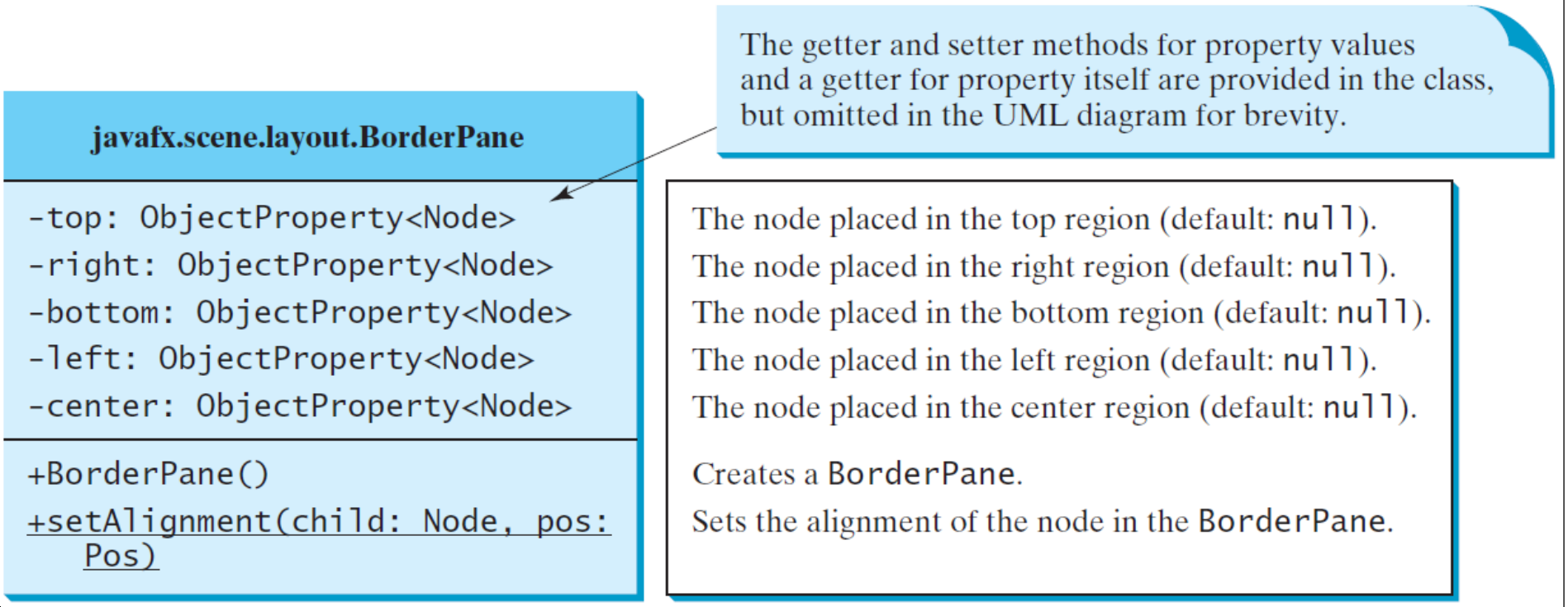
Sets the horizontal alignment for the child in the cell.

Sets the vertical alignment for the child in the cell.

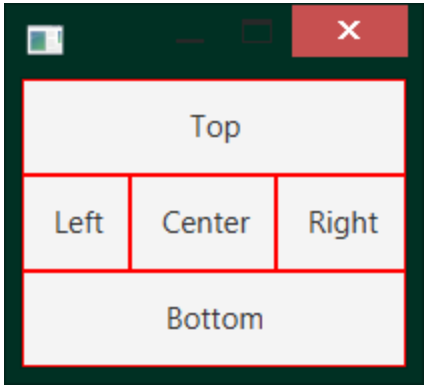
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.*;
import javafx.geometry.*;
public class ShowGridPane extends Application {
  @Override
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    GridPane pane = new GridPane();
    pane.setAlignment(Pos.CENTER);
    pane.setHgap(5.5);
    pane.setVgap(5.5);
    // Place nodes in the pane at positions   column,row
    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");
    primaryStage.setScene(scene); primaryStage.show(); }
  public static void main(String[] args) {
    launch(args);
  }}
```



BorderPane



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Label;
import javafx.geometry.Insets;
public class ShowBorderPane extends Application {
  @Override
  public void start(Stage primaryStage) {
    BorderPane pane = new BorderPane();
    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"));
    Scene scene = new Scene(pane);
    primaryStage.setScene(scene); primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
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 and VBox

javafx.scene.layout.HBox

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

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

javafx.scene.layout.VBox

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

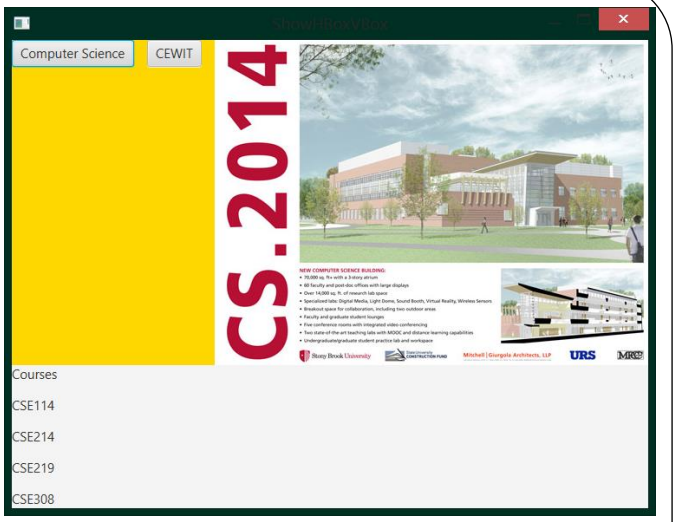
+VBox()
+VBox(spacing: double)
+setMargin(node: Node, value: Insets): void

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 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).

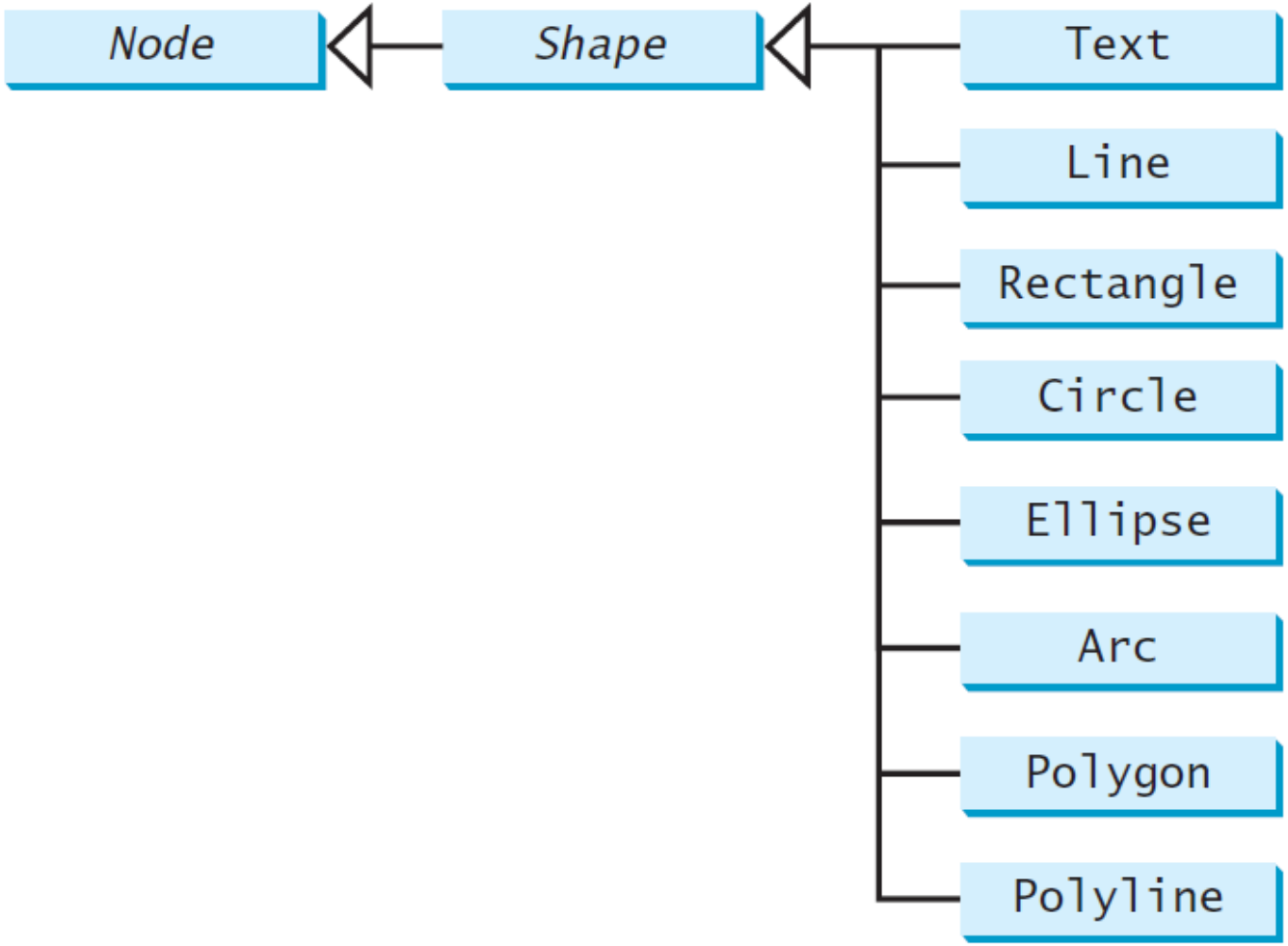
Creates a default `VBox`.
Creates a `VBox` with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
public class ShowHBoxVBox extends Application {
  @Override
  public void start(Stage primaryStage) {
    BorderPane pane = new BorderPane();
    HBox hBox = new HBox(15);
    hBox.setStyle("-fx-background-color: gold");
    hBox.getChildren().add(new Button("Computer Science"));
    hBox.getChildren().add(new Button("CEWIT"));
    ImageView imageView = new ImageView(new Image("cs14.jpg"));
    hBox.getChildren().add(imageView);
    pane.setTop(hBox);
    VBox vBox = new VBox(15);
    vBox.getChildren().add(new Label("Courses"));
    Label[] courses = {new Label("CSE114"), new Label("CSE214"),
        new Label("CSE219"), new Label("CSE308")};
    for (Label course: courses) {
      vBox.getChildren().add(course);
    }
    pane.setLeft(vBox);
    Scene scene = new Scene(pane); primaryStage.setScene(scene);
    primaryStage.show();
  }
```

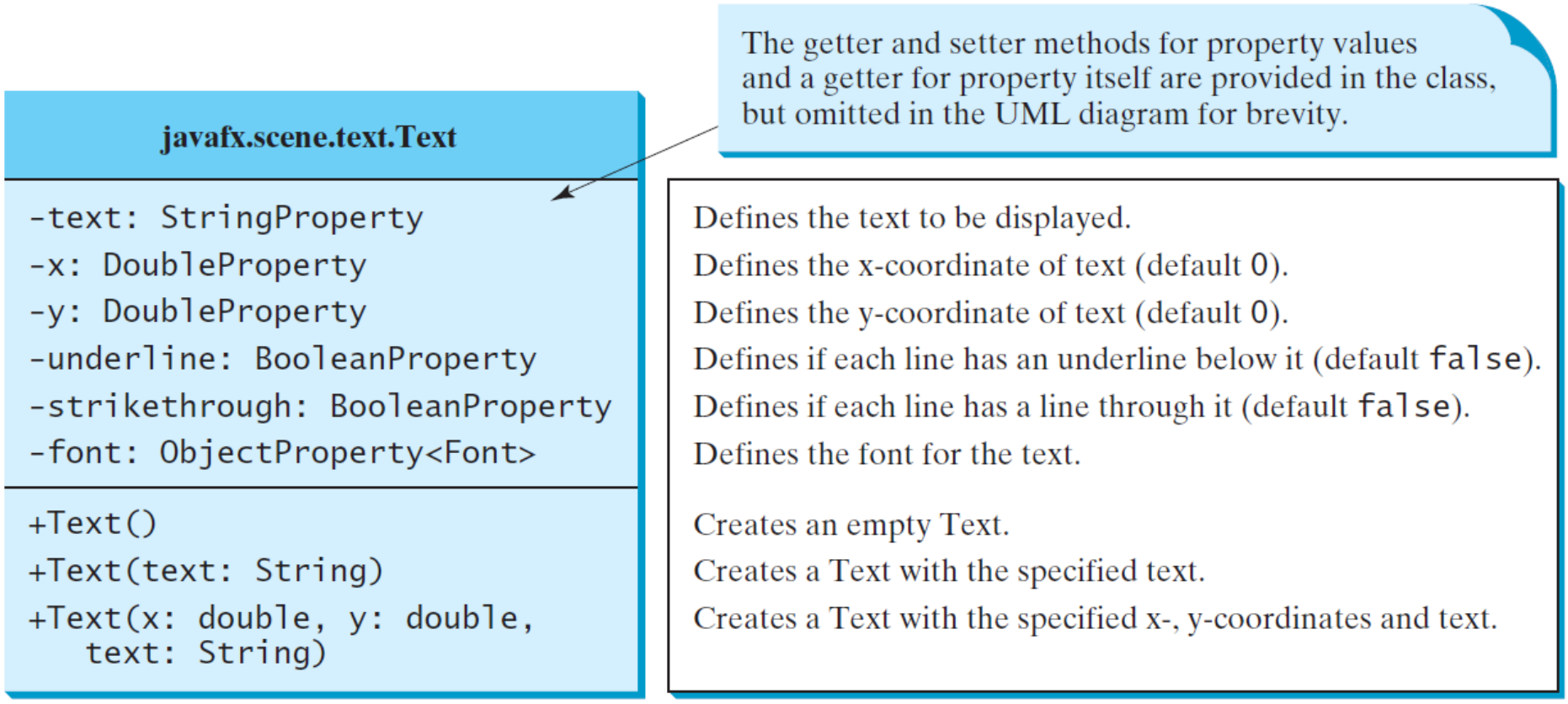


Shapes

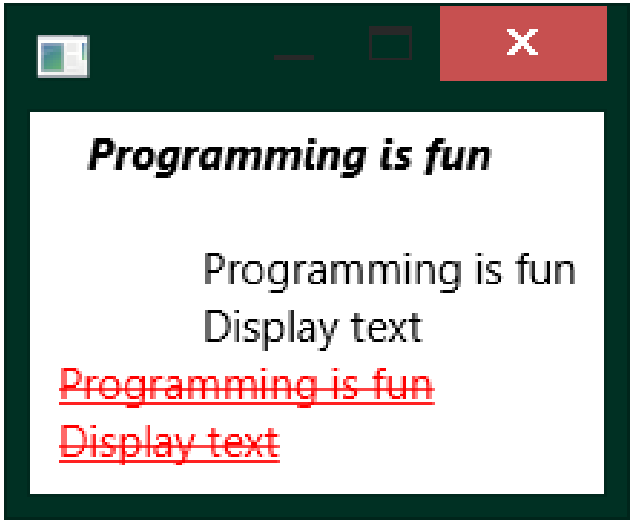
JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.

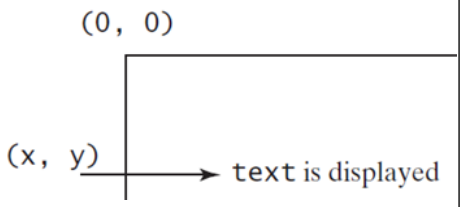


Text



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.geometry.Insets;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.FontPosture;
public class ShowText extends Application {
  @Override
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    pane.setPadding(new Insets(5, 5, 5, 5));
    Text text1 = new Text(20, 20, "Programming is fun");
    text1.setFont(Font.font("Courier", FontWeight.BOLD,
      FontPosture.ITALIC, 15));
    pane.getChildren().add(text1);
    Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
    pane.getChildren().add(text2);
    Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
    text3.setFill(Color.RED);
    text3.setUnderline(true);
    text3.setStrikethrough(true);
    pane.getChildren().add(text3);
    Scene scene = new Scene(pane, 600, 800);
    primaryStage.setScene(scene); primaryStage.show();
  }
...
}
```





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.

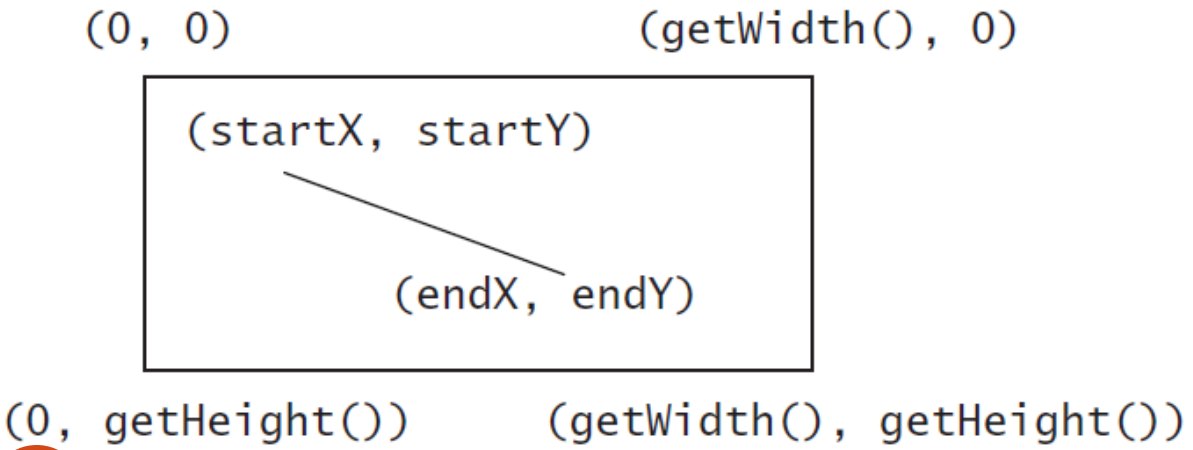
javafx.scene.shape.Line

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

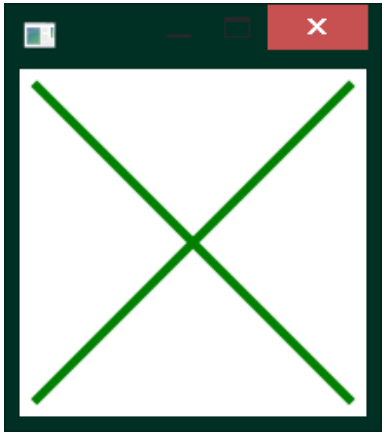
```
+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

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.



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.scene.paint.Color;
public class ShowLine extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Line line1 = new Line(10, 10, 10, 10);
        line1.endXProperty().bind(pane.widthProperty().subtract(10));
        line1.endYProperty().bind(pane.heightProperty().subtract(10));
        line1.setStrokeWidth(5);
        line1.setStroke(Color.GREEN);
        pane.getChildren().add(line1);
        Line line2 = new Line(10, 10, 10, 10);
        line2.startXProperty().bind(pane.widthProperty().subtract(10));
        line2.endYProperty().bind(pane.heightProperty().subtract(10));
        line2.setStrokeWidth(5);
        line2.setStroke(Color.GREEN);
        pane.getChildren().add(line2);
        Scene scene = new Scene(pane, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Rectangle

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.Rectangle

```
-x: DoubleProperty
-y:DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectanlge(x: double, y:
   double, width: double,
   height: double)
```

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).

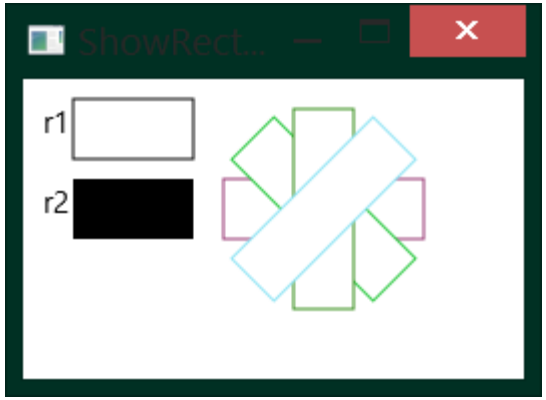
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

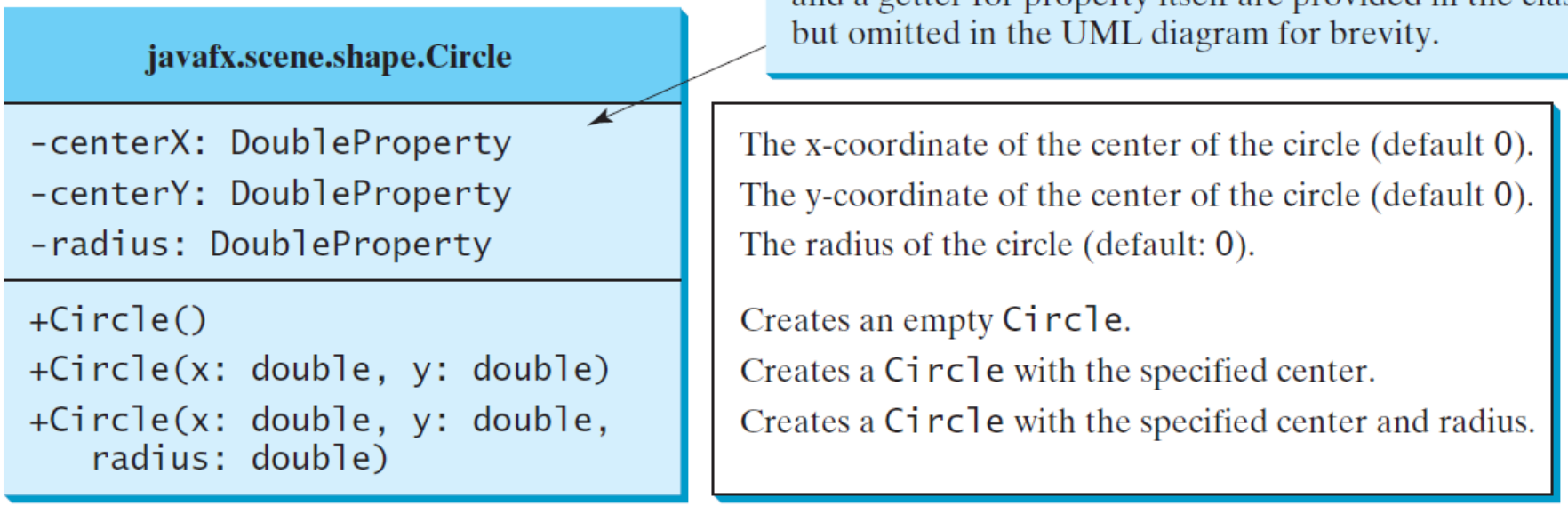
Creates an empty `Rectangle`.

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

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
import javafx.scene.shape.Rectangle;
import javafx.scene.paint.Color;
import java.util.Collections;
public class ShowRectangle extends Application {
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    Rectangle r1 = new Rectangle(25, 10, 60, 30);
    r1.setStroke(Color.BLACK);
    r1.setFill(Color.WHITE);
    pane.getChildren().add(new Text(10, 27, "r1"));
    pane.getChildren().add(r1);
    Rectangle r2 = new Rectangle(25, 50, 60, 30);
    pane.getChildren().add(new Text(10, 67, "r2"));
    pane.getChildren().add(r2);
    for (int i = 0; i < 4; i++) {
      Rectangle r = new Rectangle(100, 50, 100, 30);
      r.setRotate(i * 360 / 8);
      r.setStroke(Color.color(Math.random(), Math.random(),
          Math.random()));
      r.setFill(Color.WHITE);
      pane.getChildren().add(r);
    }
    Scene scene = new Scene(pane, 250, 150);
    primaryStage.setScene(scene); primaryStage.show();
  }
  ...// main
```



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.

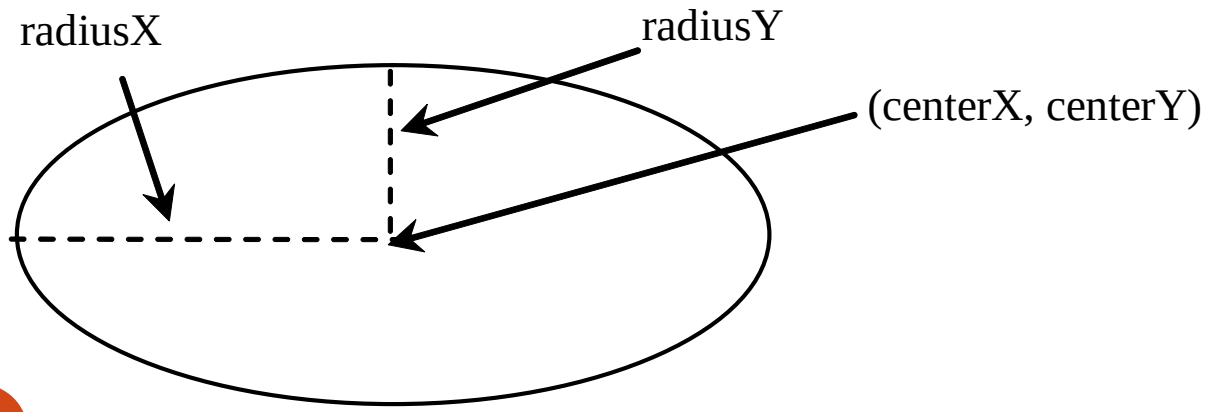
javafx.scene.shape.Ellipse

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

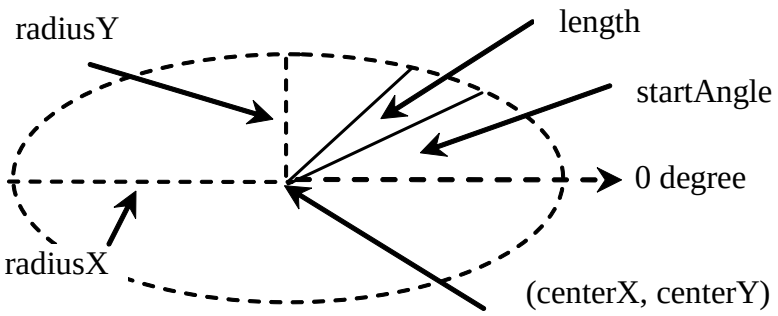
```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

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.

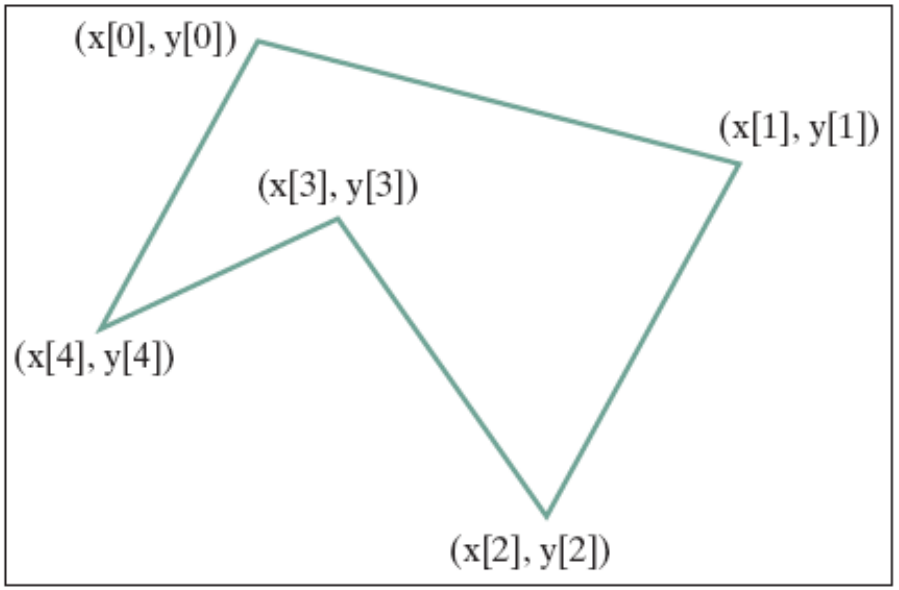
The angular extent of the arc in degrees.

The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

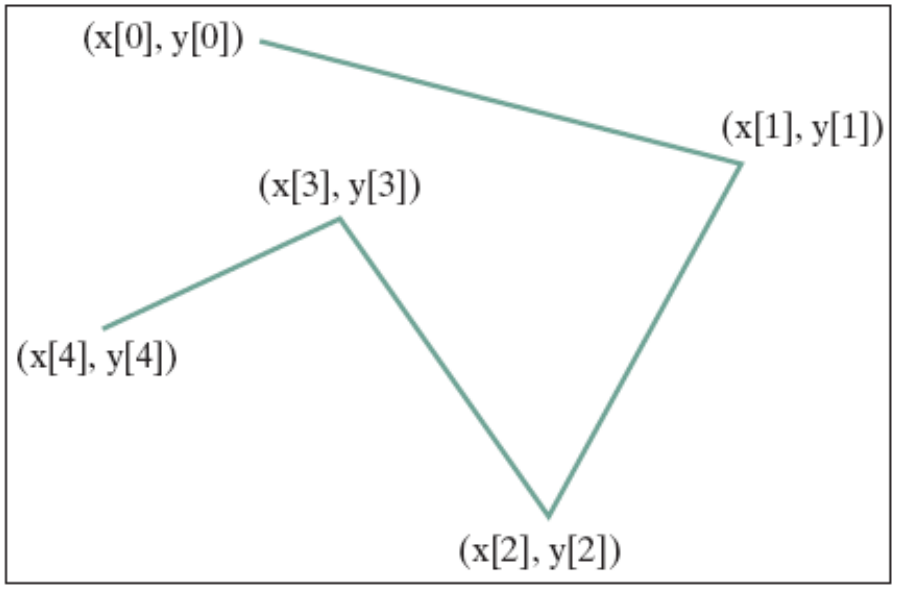
Creates an empty `Arc`.

Creates an `Arc` with the specified arguments.

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.

Event Programming

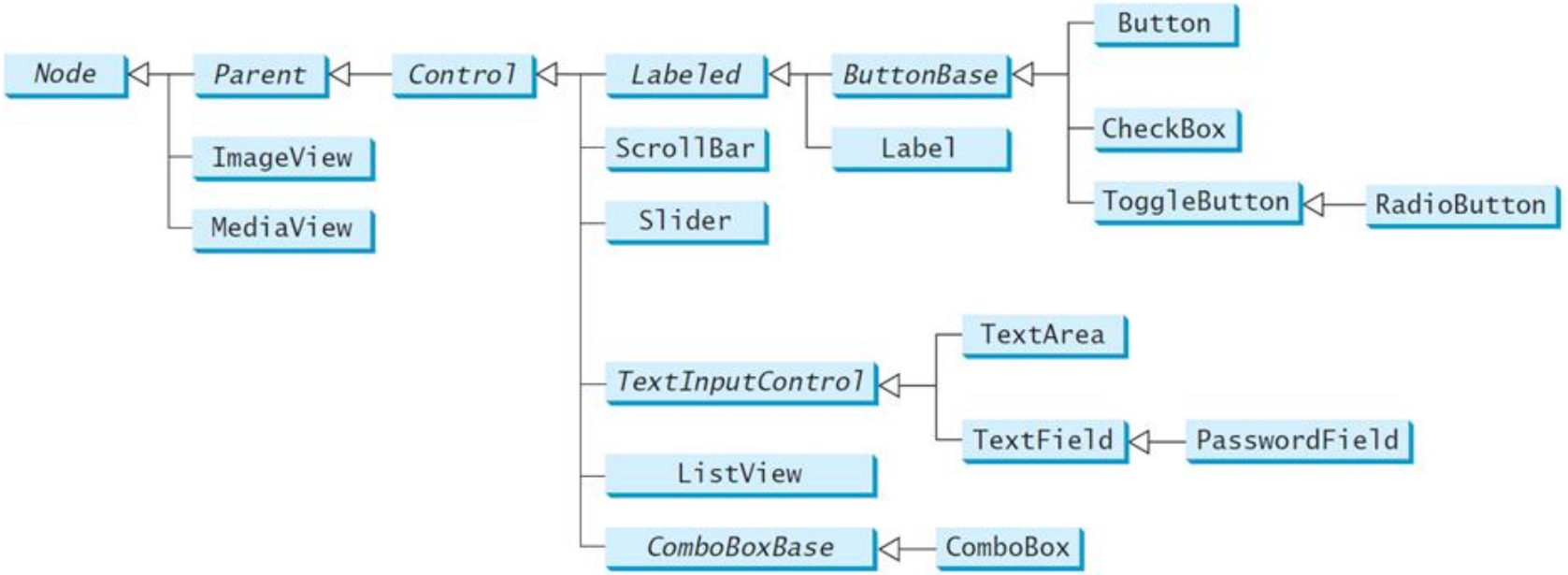
- Procedural programming is executed in procedural/statement order
- In event-driven programming, code is executed upon activation of events
- Operating Systems constantly monitor events
  - Ex: keystrokes, mouse clicks, etc…
- The OS:
  - sorts out these events
  - reports them to the appropriate programs

Where do we come in?

- For each control (button, combo box, etc.):
  - define an event handler
  - construct an instance of event handler
  - tell the control who its event handler is
- Event Handler?
  - code with response to event
  - a.k.a. event listener

Java's Event Handling

- An *event source* is a GUI control
  - JavaFX: **`Button`**, **`ChoiceBox`**, **`ListView`**, etc.



http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

- different types of sources:
  - can detect different types of events
  - can register different types of listeners (handlers)

Java's Event Handling

- When the user interacts with a control (source):
  - an *event object* is constructed
  - the event object is sent to all registered *listener objects*
  - the listener object (handler) responds as you defined it to

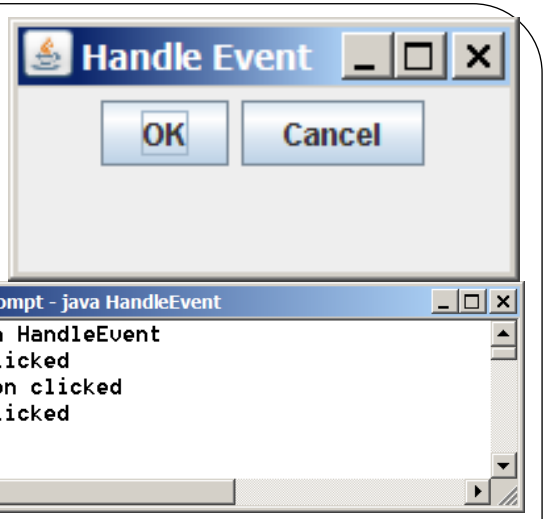
Event Listeners (Event Handler)

- Defined by you, the application programmer
  - you customize the response
  - How?
    - Inheritance & Polymorphism
- You define your own listener class
  - implement the appropriate interface
  - define responses in all necessary methods

Event Objects

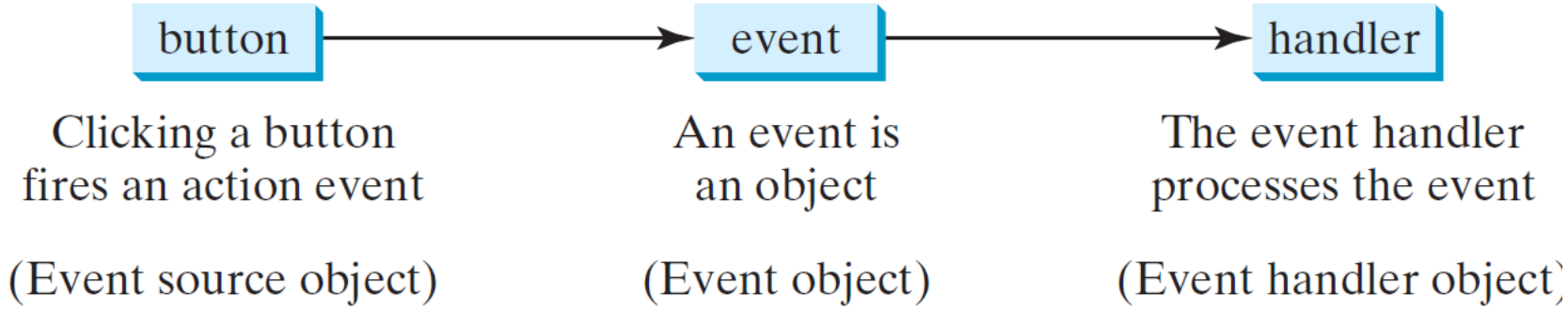
- Contain information about the event
- Like what?
  - location of mouse click
  - event source that was interacted with
  - etc.
- Listeners use them to properly respond
  - different methods inside a listener object can react differently to different types of interactions

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
public class HandleEvent extends Application {
 public void start(Stage primaryStage) {
        HBox pane = new HBox(10);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        pane.getChildren().addAll(btOK, btCancel);
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);  primaryStage.show();
 }…/*main*/}
class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }}
class CancelHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Cancel button clicked");
    }}
```

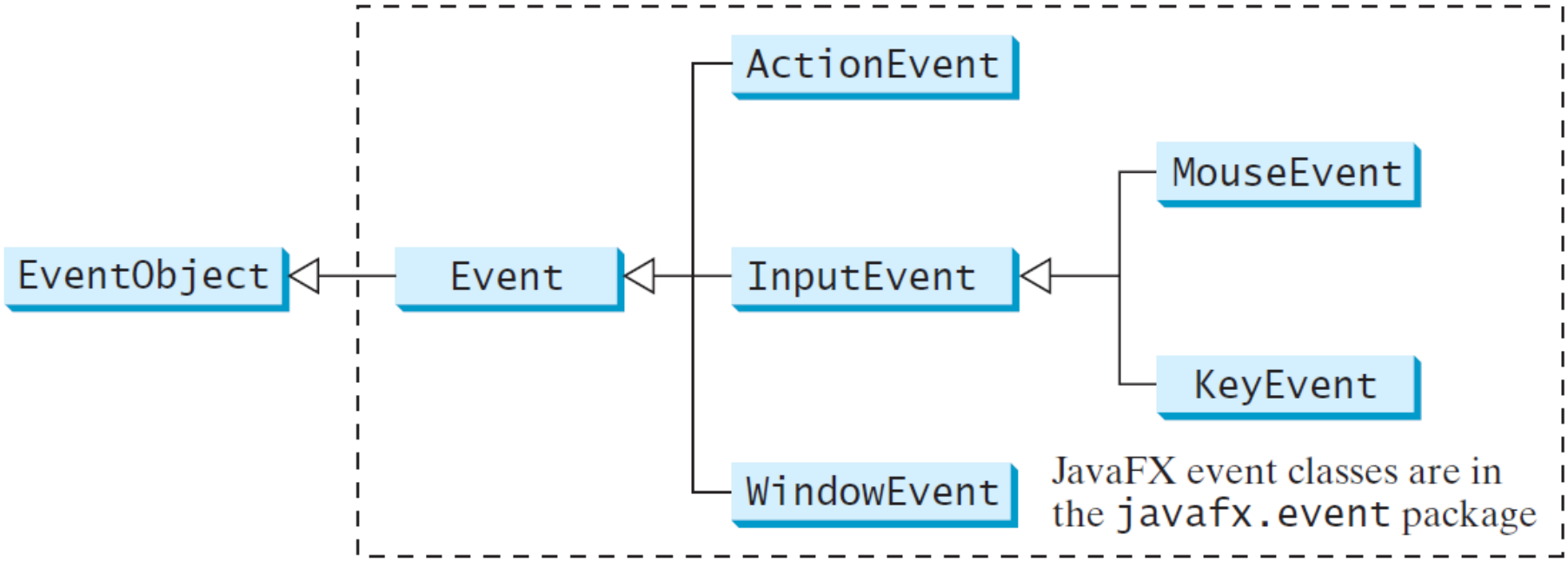


Handling GUI Events

- Source object: button.
  - An event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes.
- An event can be defined as a type of signal to the program that something has happened.
- Listener object contains a method for processing the event.



Event Classes



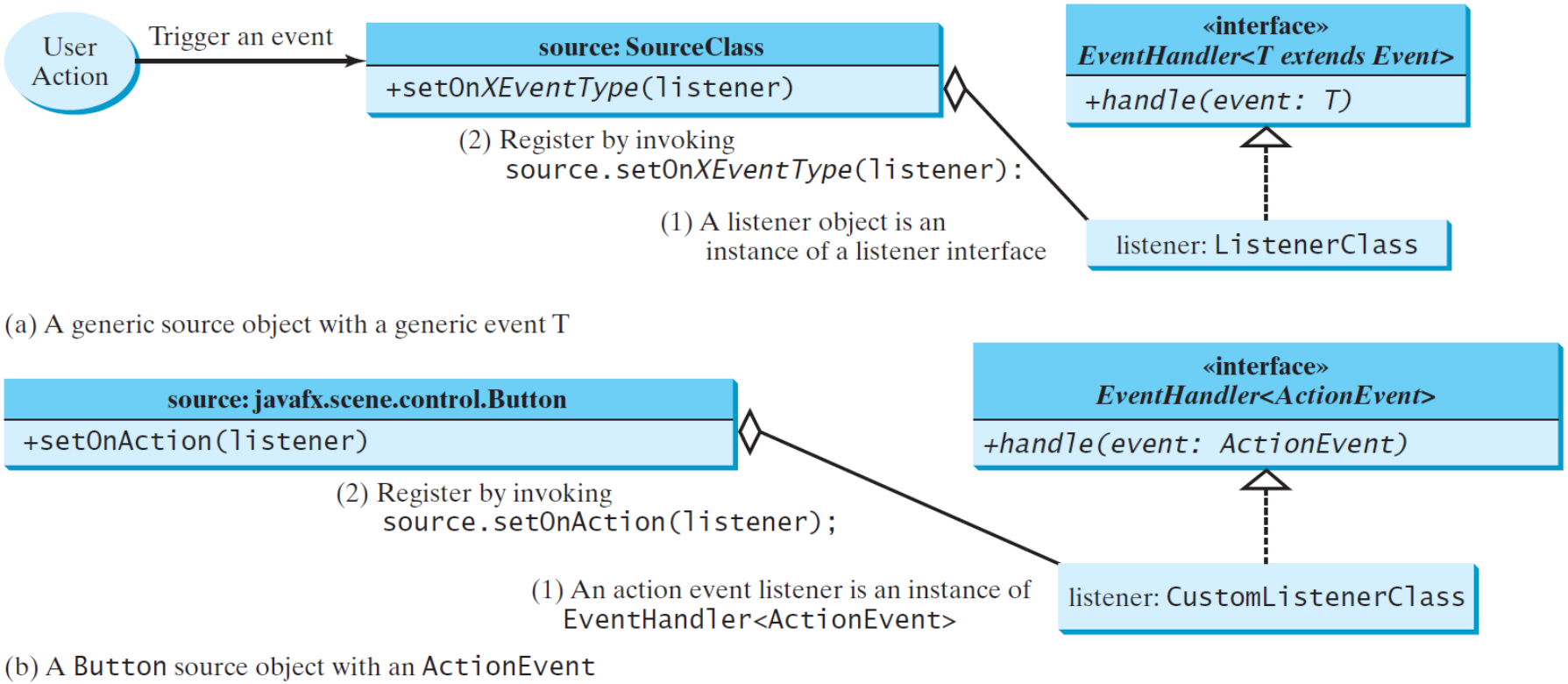
Event Information

- An event object contains whatever properties are pertinent to the event:
  - the ***source object*** of the event using the **`getSource()`** instance method in the **`EventObject`** class.
- The subclasses of **`EventObject`** deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes.

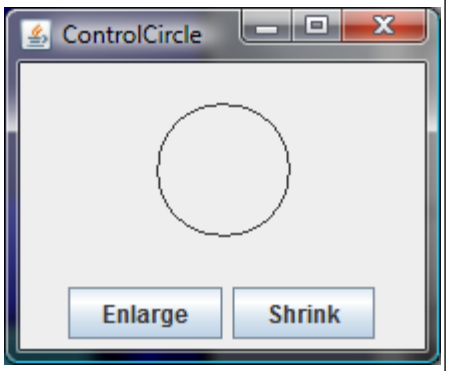
Selected User Actions and Handlers

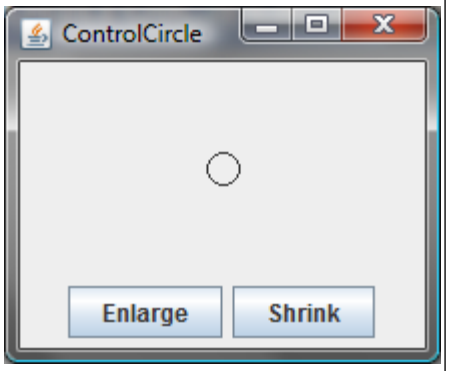
User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

The Delegation Model



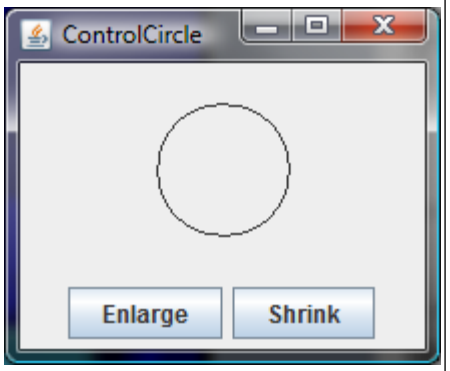
ControlCircle program that uses two buttons to control the size of a circle

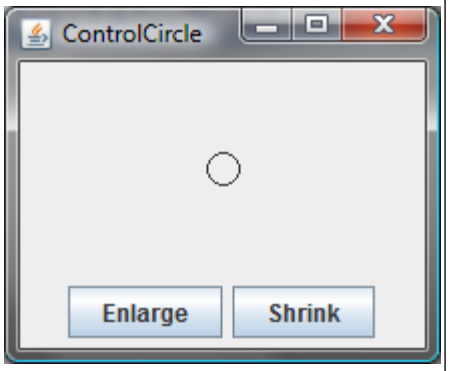





```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
public class ControlCircle extends Application {
    private CirclePane circlePane = new CirclePane();
    @Override
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btEnlarge = new Button("Enlarge");
        Button btShrink = new Button("Shrink");
        hBox.getChildren().add(btEnlarge);
        hBox.getChildren().add(btShrink);
        btEnlarge.setOnAction(new EnlargeHandler());
        BorderPane borderPane = new BorderPane();
        borderPane.setCenter(circlePane);
        borderPane.setBottom(hBox);
        BorderPane.setAlignment(hBox, Pos.CENTER);
        Scene scene = new Scene(borderPane, 200, 150);
        primaryStage.setScene(scene); primaryStage.show();
    }
```

ControlCircle program that uses two buttons to control the size of a circle






```
        // Inner Class
        class EnlargeHandler
            implements EventHandler<ActionEvent> {
            @Override
            public void handle(ActionEvent e) {
                circlePane.enlarge();
            }
        }
    }
    class CirclePane extends StackPane {
        private Circle circle = new Circle(50);
        public CirclePane() {
            getChildren().add(circle);
            circle.setStroke(Color.BLACK);
            circle.setFill(Color.WHITE);
        }
        public void enlarge() {
            circle.setRadius(circle.getRadius() + 2);
        }
        public void shrink() {
            circle.setRadius(circle.getRadius() > 2
                    ? circle.getRadius() - 2 : circle.getRadius());
        }
    }
```

Inner Class Listeners

- A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).
  - Any object instance of the inner handler class has access to all GUI fields of the outer class.
  - It will not be shared by other applications.

Inner Classes

```
public class OuterClass {
  private int data = 0;
  OuterClass(){
    InnerClass y = new InnerClass();
    y.m2();
  }
  public void m1() {
    data++;
  }
  public static void main(String[] args) {
    OuterClass x = new OuterClass();
    System.out.println(x.data);
  }
  class InnerClass {
    public void m2() {
      /* Directly reference data and
        method defined in outer class */
      data++;
      m1();
    }
  }
}
```

- The **`InnerClass`** is a member of **`OuterClass`**
  - An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.
  - An inner class is compiled into a class named

`OuterClass$InnerClass.class`

Inner Classes

- An inner class can be declared **`public`**, **`protected`**, or **`private`** subject to the same visibility rules applied to a member of the class.
- An inner class can be declared **`static`**:
  - The **`static`** inner class can be accessed using the outer class name,
  - However, a **`static`** inner class cannot access nonstatic members of the outer class.

Anonymous Inner Classes

- Inner class listeners can be shortened using anonymous inner classes: inner classes without a name.
  - It combines declaring an inner class and creating an instance of the class in one step.
- An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {
 // Implement or override methods in superclass/interface
 // Other methods if necessary
}
```

Anonymous Inner Classes

- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance.
- If an anonymous inner class implements an interface, the constructor is **`Object()`**.
- An anonymous inner class is compiled into a class named **`OuterClassName$n.class`**, where **`n`** is the count of inner classes.

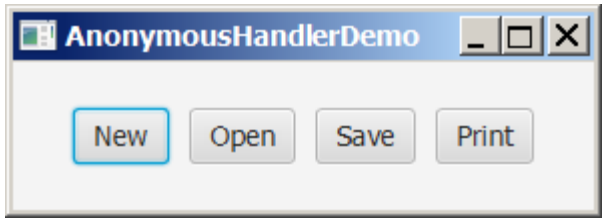
Anonymous Inner Classes

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
}
```

(a) Inner class EnlargeListener

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new class EnlargeHandler
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
public class AnonymousHandlerDemo extends Application {
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btNew = new Button("New");
        Button btOpen = new Button("Open"); //btSave, btPrint btns.
        hBox.getChildren().addAll(btNew, btOpen);
        // Create and register the handler
        btNew.setOnAction(new EventHandler<ActionEvent>() {
            @Override // Override the handle method
            public void handle(ActionEvent e) {
                System.out.println("Process New");
            }
        });
        btOpen.setOnAction(new EventHandler<ActionEvent>() {
            @Override // Override the handle method
            public void handle(ActionEvent e) {
                System.out.println("Process Open");
            }
        });
```

```
        Scene scene = new Scene(hBox, 300, 50);
        primaryStage.setTitle("AnonymousHandlerDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Simplifying Event Handing Using Lambda Expressions

- *Lambda expression* is a new feature in Java 8.
  - Predefined functions for the type of the input.
- Lambda expressions can be viewed as an anonymous method with a concise syntax.

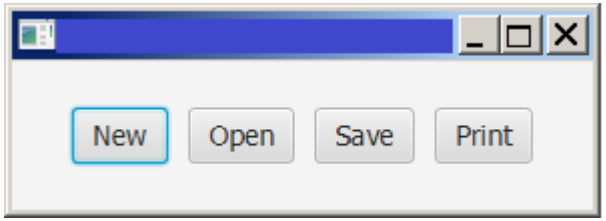
```
btEnlarge.setOnAction(
  new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      // Code for processing event e
    }
  }
});
```

(a) Anonymous inner class event handler

```
btEnlarge.setOnAction(e -> {
  // Code for processing event e
});
```

(b) Lambda expression event handler

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
public class LambdaHandlerDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Hold two buttons in an HBox
        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
        Button btNew = new Button("New");
        Button btOpen = new Button("Open");
        Button btSave = new Button("Save");
        Button btPrint = new Button("Print");
        hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
        btNew.setOnAction(e -> {System.out.println("Process New");});
        btOpen.setOnAction(e -> {System.out.println("Process Open");});
        btSave.setOnAction(e -> {System.out.println("Process Save");});
        btPrint.setOnAction(e -> {System.out.println("Process Print");});
        Scene scene = new Scene(hBox, 300, 50);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);   }}
```



Output:
Process New
Process Open
Process Save
Process Print

Basic Syntax for a Lambda Expression

- The basic syntax for a lambda expression is either:

```
(type1 param1, type2 param2, ...) -> expression
```

or

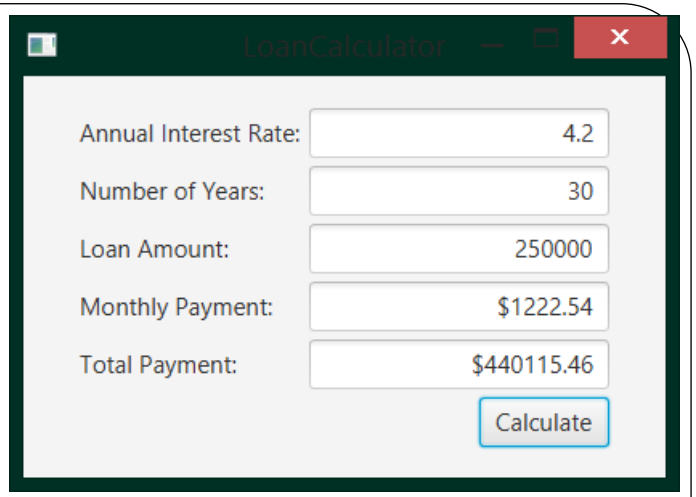
```
(type1 param1, type2 param2, ...) -> { statements; }
```

  - The data type for a parameter may be explicitly declared or implicitly inferred by the compiler.
  - The parentheses can be omitted if there is only one parameter without an explicit data type.

Single Abstract Method Interface (SAM)

- The statements in the lambda expression is all for that method.
  - If it contains multiple methods, the compiler will not be able to compile the lambda expression.
  - So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
  - Such an interface is known as a *functional interface*, or a *Single Abstract Method* (SAM) interface.

Loan Calculator



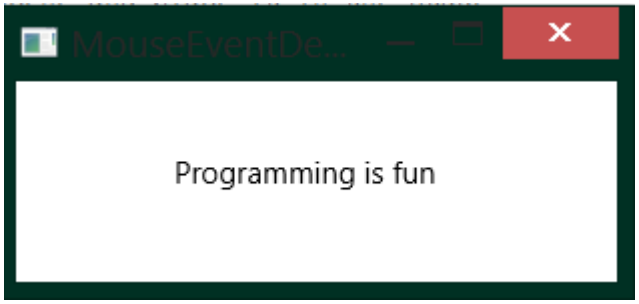

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Pos;
import javafx.geometry.HPos;
public class LoanCalculator extends Application {
    private TextField tfAnnualInterestRate = new TextField();
    private TextField tfNumberOfYears = new TextField();
    private TextField tfLoanAmount = new TextField();
    private TextField tfMonthlyPayment = new TextField();
    private TextField tfTotalPayment = new TextField();
    private Button btCalculate = new Button("Calculate");
    @Override
    public void start(Stage primaryStage) {
        // Create UI
        GridPane gridPane = new GridPane();
        gridPane.setHgap(5);
        gridPane.setVgap(5);
        gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
        gridPane.add(tfAnnualInterestRate, 1, 0);
        gridPane.add(new Label("Number of Years:"), 0, 1);
        gridPane.add(tfNumberOfYears, 1, 1);
        gridPane.add(new Label("Loan Amount:"), 0, 2);
        gridPane.add(tfLoanAmount, 1, 2);
        gridPane.add(new Label("Monthly Payment:"), 0, 3);
        gridPane.add(tfMonthlyPayment, 1, 3);
        gridPane.add(new Label("Total Payment:"), 0, 4);
        gridPane.add(tfTotalPayment, 1, 4);
        gridPane.add(btCalculate, 1, 5);
```

```
        btCalculate.setOnAction(e -> calculateLoanPayment());
        Scene scene = new Scene(gridPane, 400, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    private void calculateLoanPayment() {
        // Get values from text fields
        double interest = Double.parseDouble(tfAnnualInterestRate.getText());
        int year = Integer.parseInt(tfNumberOfYears.getText());
        double loanAmount = Double.parseDouble(tfLoanAmount.getText());
        // Create a loan object
        Loan loan = new Loan(interest, year, loanAmount);
        // Display monthly payment and total payment
        tfMonthlyPayment.setText(String.format("$%.2f", loan.getMonthlyPayment()));
        tfTotalPayment.setText(String.format("$%.2f", loan.getTotalPayment()));
    }
    public static void main(String[] args) {
        launch(args);
    }
}
class Loan implements java.io.Serializable {
    private double annualInterestRate;
    private int numberOfYears;
    private double loanAmount;
    private java.util.Date loanDate;
    public Loan(double annualInterestRate, int numberOfYears, double loanAmount) {
        this.annualInterestRate = annualInterestRate;
        this.numberOfYears = numberOfYears;
        this.loanAmount = loanAmount;
        loanDate = new java.util.Date();
    }
    public double getAnnualInterestRate() {
        return annualInterestRate;
    }
    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }
```

```
    public int getNumberOfYears() {
        return numberOfYears;
    }
    public void setNumberOfYears(int numberOfYears) {
        this.numberOfYears = numberOfYears;
    }
    public double getLoanAmount() {
        return loanAmount;
    }
    public void setLoanAmount(double loanAmount) {
        this.loanAmount = loanAmount;
    }
    public double getMonthlyPayment() {
        double monthlyInterestRate = annualInterestRate / 1200;
        double monthlyPayment = loanAmount * monthlyInterestRate / (1
                - (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
        return monthlyPayment;
    }
    public double getTotalPayment() {
        double totalPayment = getMonthlyPayment() * numberOfYears * 12;
        return totalPayment;
    }
    public java.util.Date getLoanDate() {
        return loanDate;
    }
}
```

MouseEvent

javafx.scene.input.MouseEvent	
+getButton(): MouseButton	Indicates which mouse button has been clicked.
+getClickCount(): int	Returns the number of mouse clicks associated with this event.
+getX(): double	Returns the *x*-coordinate of the mouse point in the event source node.
+getY(): double	Returns the *y*-coordinate of the mouse point in the event source node.
+getSceneX(): double	Returns the *x*-coordinate of the mouse point in the scene.
+getSceneY(): double	Returns the *y*-coordinate of the mouse point in the scene.
+getScreenX(): double	Returns the *x*-coordinate of the mouse point in the screen.
+getScreenY(): double	Returns the *y*-coordinate of the mouse point in the screen.
+isAltDown(): boolean	Returns true if the `Alt` key is pressed on this event.
+isControlDown(): boolean	Returns true if the `Control` key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse `Meta` button is pressed on this event.
+isShiftDown(): boolean	Returns true if the `Shift` key is pressed on this event.




```
// Move the text with the mouse clicked
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
public class MouseEventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Text text = new Text(20, 20, "Programming is fun");
        pane.getChildren().add(text);
        text.setOnMouseDragged(e -> {
            text.setX(e.getX());
            text.setY(e.getY());
        });
        Scene scene = new Scene(pane, 300, 100);
        primaryStage.setTitle("MouseEventDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

The `KeyEvent` Class

javafx.scene.input.KeyEvent	
+getCharacter(): String	Returns the character associated with the key in this event.
+getCode(): KeyCode	Returns the key code associated with the key in this event.
+getText(): String	Returns a string describing the key code.
+isAltDown(): boolean	Returns true if the `Alt` key is pressed on this event.
+isControlDown(): boolean	Returns true if the `Control` key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse `Meta` button is pressed on this event.
+isShiftDown(): boolean	Returns true if the `Shift` key is pressed on this event.

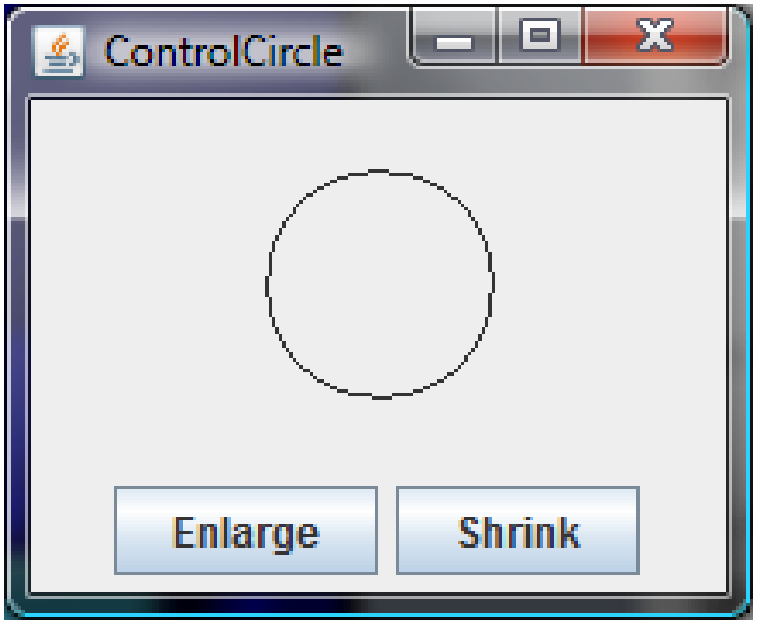
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
public class KeyEventDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    Text text = new Text(20, 20, "A");
    text.setFocusTraversable(true);
    pane.getChildren().add(text);
    text.setOnKeyPressed(e -> {
      switch (e.getCode()) {
        case DOWN: text.setY(text.getY() + 10); break;
        case UP:  text.setY(text.getY() - 10); break;
        case LEFT: text.setX(text.getX() - 10); break;
        case RIGHT: text.setX(text.getX() + 10); break;
        default:
          if (Character.isLetterOrDigit(e.getText().charAt(0)))
            text.setText(e.getText());
      }
    });
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("KeyEventDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
      launch(args);
  }
}
```

The `KeyCode` Constants

Constant	*Description*	*Constant*	*Description*
`HOME`	The Home key	`CONTROL`	The Control key
`END`	The End key	`SHIFT`	The Shift key
`PAGE_UP`	The Page Up key	`BACK_SPACE`	The Backspace key
`PAGE_DOWN`	The Page Down key	`CAPS`	The Caps Lock key
`UP`	The up-arrow key	`NUM_LOCK`	The Num Lock key
`DOWN`	The down-arrow key	`ENTER`	The Enter key
`LEFT`	The left-arrow key	`UNDEFINED`	The `keyCode` unknown
`RIGHT`	The right-arrow key	`F1` to `F12`	The function keys from F1 to F12
`ESCAPE`	The Esc key	`0` to `9`	The number keys from 0 to 9
`TAB`	The Tab key	`A` to `Z`	The letter keys from A to Z

Control Circle with Mouse and Key

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.input.KeyCode;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.HBox;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class ControlCircleWithMouseAndKey extends Application {
  private CirclePane circlePane = new CirclePane();

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Hold two buttons in an HBox
    HBox hBox = new HBox();
    hBox.setSpacing(10);
    hBox.setAlignment(Pos.CENTER);
    Button btEnlarge = new Button("Enlarge");
    Button btShrink = new Button("Shrink");
    hBox.getChildren().add(btEnlarge);
    hBox.getChildren().add(btShrink);

    // Create and register the handler
    btEnlarge.setOnAction(e -> circlePane.enlarge());
    btShrink.setOnAction(e -> circlePane.shrink());

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(circlePane);
    borderPane.setBottom(hBox);
    BorderPane.setAlignment(hBox, Pos.CENTER);
```



```
    // Create a scene and place it in the stage
    Scene scene = new Scene(borderPane, 200, 150);
    primaryStage.setTitle("ControlCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

    circlePane.setOnMouseClicked(e -> {
      if (e.getButton() == MouseButton.PRIMARY) {
        circlePane.enlarge();
      }
      else if (e.getButton() == MouseButton.SECONDARY) {
        circlePane.shrink();
      }
    });

    scene.setOnKeyPressed(e -> {
      if (e.getCode() == KeyCode.UP) {
        circlePane.enlarge();
      }
      else if (e.getCode() == KeyCode.DOWN) {
        circlePane.shrink();
      }
    });
  }

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

Listeners for **`Observable`** Objects

- You can add a listener to process a value change in an observable object: an instance of **`javafx.beans.Observable`**
  - Every binding property is an instance of **`Observable`**.
  - **`Observable`** contains the **`addListener(InvalidationListener listener)`** method for adding a listener.
  - Once the value is changed in the property, a listener is notified.
  - The listener class should implement the **`InvalidationListener`** interface, which uses the **`invalidated(Observable o)`** method to handle the property value change.

Listeners for **Observable** Objects

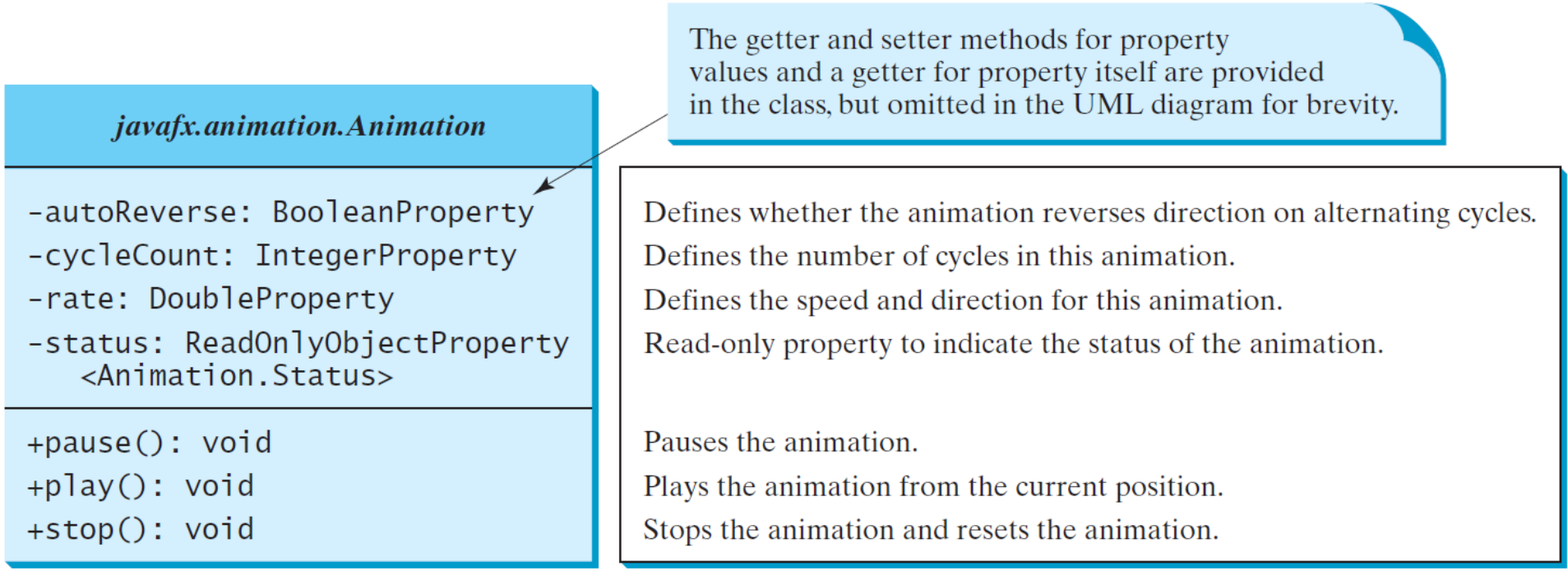
```
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
public class ObservablePropertyDemo {
  public static void main(String[] args) {
    DoubleProperty balance = new SimpleDoubleProperty();
    balance.addListener(new InvalidationListener() {
      public void invalidated(Observable ov) {
        System.out.println("The new value is " +
          balance.doubleValue());
      }
    });
    balance.set(4.5);
  }
}
```

Output:

The new value is 4.5

Animation

- JavaFX provides the **`Animation`** class with the core functionality for all animations:



PathTransition

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.animation.PathTransition

```
-duration: ObjectProperty<Duration>
-node: ObjectProperty<Node>
-orientation: ObjectProperty
   <PathTransition.OrientationType>
-path: ObjectType<Shape>
```

```
+PathTransition()
+PathTransition(duration: Duration,
   path: Shape)
+PathTransition(duration: Duration,
   path: Shape, node: Node)
```

The duration of this transition.

The target node of this transition.

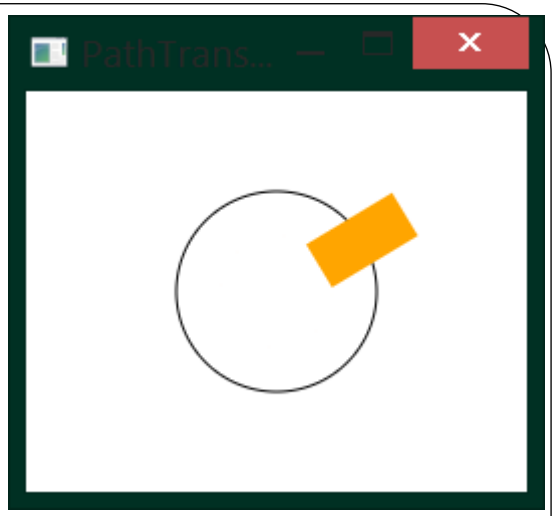
The orientation of the node along the path.

The shape whose outline is used as a path to animate the node move.

Creates an empty `PathTransition`.

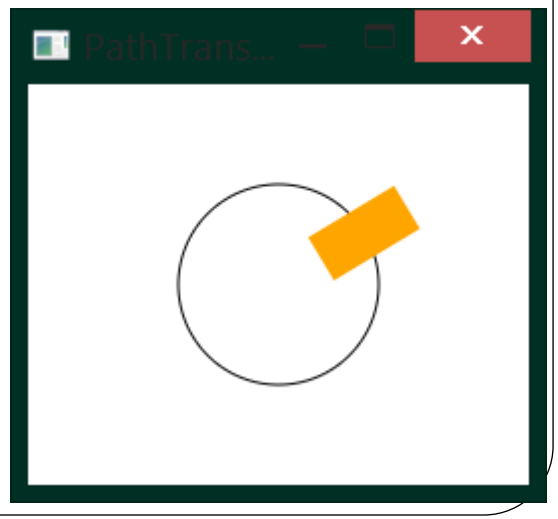
Creates a `PathTransition` with the specified duration and path.

Creates a `PathTransition` with the specified duration, path, and node.




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Circle;
import javafx.animation.PathTransition;
import javafx.animation.Timeline;
import javafx.util.Duration;
public class PathTransitionDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Rectangle rectangle = new Rectangle(0, 0, 25, 50);
        rectangle.setFill(Color.ORANGE);
        Circle circle = new Circle(125, 100, 50);
        circle.setFill(Color.WHITE);
        circle.setStroke(Color.BLACK);
        pane.getChildren().addAll(circle,rectangle);
        // Create a path transition
        PathTransition pt = new PathTransition();
        pt.setDuration(Duration.millis(4000));
        pt.setPath(circle);
        pt.setNode(rectangle);
```

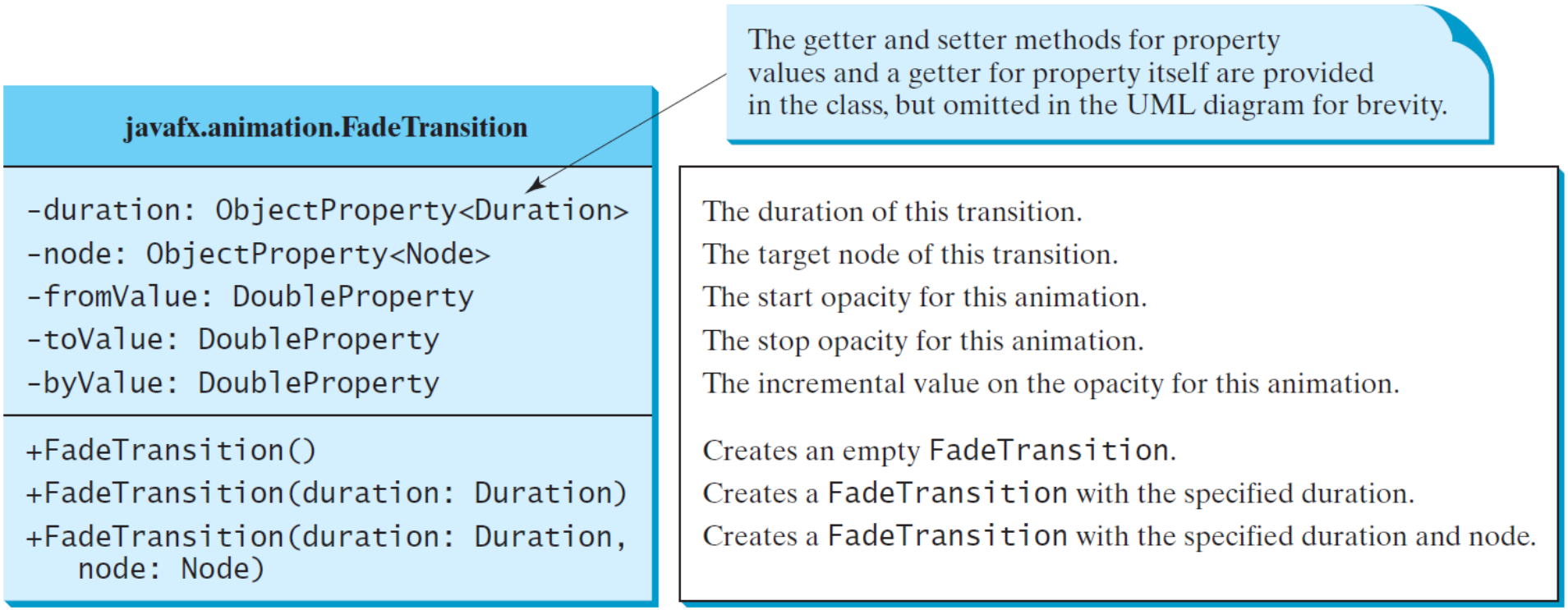
```
    pt.setOrientation(
      PathTransition.OrientationType.
        ORTHOGONAL_TO_TANGENT);
    pt.setCycleCount(Timeline.INDEFINITE);
    pt.setAutoReverse(true);
    pt.play(); // Start animation
    circle.setOnMousePressed(e -> pt.pause());
    circle.setOnMouseReleased(e -> pt.play());
    Scene scene = new Scene(pane, 250, 200);
    primaryStage.setTitle("PathTransitionDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void
          main(String[] args){
      launch(args);
  }
}
```

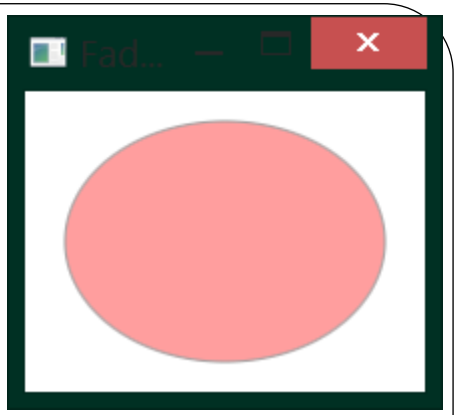


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.animation.PathTransition;
import javafx.scene.image.ImageView;
import javafx.util.Duration;
public class FlagRisingAnimation extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        ImageView imageView = new ImageView("us.jpg");
        pane.getChildren().add(imageView);
        PathTransition pt = new PathTransition(
            Duration.millis(10000),
              new Line(100, 200, 100, 0),
            imageView);
        pt.setCycleCount(5);
        pt.play(); // Start animation
        Scene scene = new Scene(pane, 250, 200);
        primaryStage.setScene(scene); primaryStage.show();
    }
```

FadeTransition

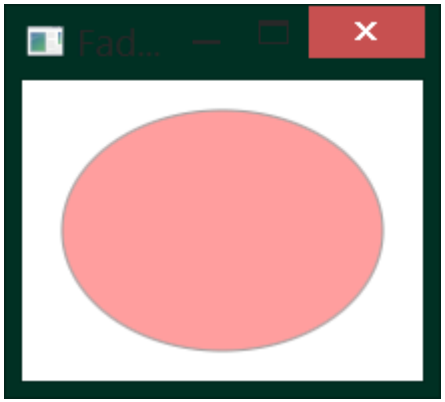
The **`FadeTransition`** class animates the change of the opacity in a node over a given time:





```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.util.Duration;
public class FadeTransitionDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Ellipse ellipse = new Ellipse(10, 10, 100, 50);
        ellipse.setFill(Color.RED);
        ellipse.setStroke(Color.BLACK);
        ellipse.centerXProperty().bind(pane.widthProperty().divide(2));
        ellipse.centerYProperty().bind(pane.heightProperty().divide(2));
        ellipse.radiusXProperty().bind(pane.widthProperty().multiply(0.4));
        ellipse.radiusYProperty().bind(pane.heightProperty().multiply(0.4));
        pane.getChildren().add(ellipse);
        // Apply a fade transition to ellipse
        FadeTransition ft = new FadeTransition(Duration.millis(3000), ellipse);
        ft.setFromValue(1.0);
        ft.setToValue(0.1);
        ft.setCycleCount(Timeline.INDEFINITE);
        ft.setAutoReverse(true);
        ft.play(); // Start animation
        // Control animation
        ellipse.setOnMousePressed(e -> ft.pause());
        ellipse.setOnMouseReleased(e -> ft.play());
```

```
    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 150);
    primaryStage.setTitle("FadeTransitionDemo"); // 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);
  }
}
```



Timeline

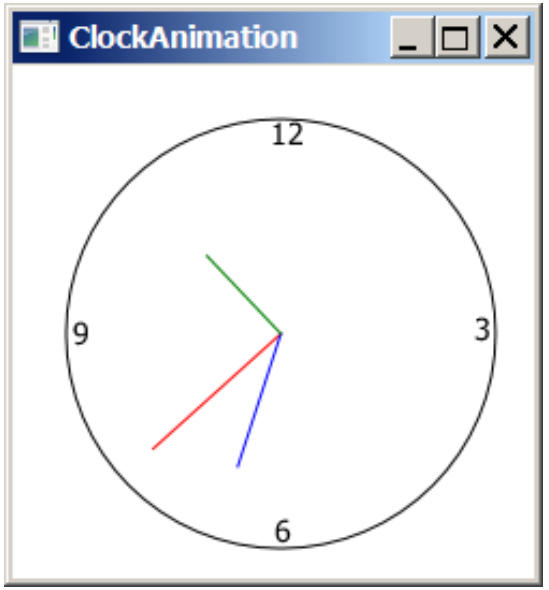
- **`PathTransition`** and **`FadeTransition`** define specialized animations.
- The **`javafx.animation.Timeline`** class can be used to program any animation using one or more **`javafx.animation.KeyFrames`**
  - **`KeyFrame`** defines target values at a specified point in time for a set of variables that are interpolated along a **`Timeline`**.
    - Each **`KeyFrame`** is executed sequentially at a specified time interval.
  - **`Timeline`** inherits from **`Animation`**.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.util.Duration;
public class TimelineDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Text text = new Text(20, 50, "Programming if fun");
        text.setFill(Color.RED);
        pane.getChildren().add(text);
        // Create a handler for changing text
        EventHandler<ActionEvent> eH = e -> {
            if (text.getText().length() != 0) {
                text.setText("");
            } else {
                text.setText("Programming is fun");
            }
        };
        Timeline animation = new Timeline(new KeyFrame(Duration.millis(500), eH));
        animation.setCycleCount(Timeline.INDEFINITE);
        // Start animation
        animation.play();
```

```
        // Pause and resume animation
        text.setOnMouseClicked(e -> {
            if (animation.getStatus() ==
                        Animation.Status.PAUSED) {
                animation.play();
            } else {
                animation.pause();
            }
        });
        Scene scene = new Scene(pane, 250, 50);
        primaryStage.setTitle("TimelineDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Clock Animation

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.util.Duration;
public class ClockAnimation extends Application {
  @Override
  public void start(Stage primaryStage) {
    ClockPane clock = new ClockPane(); // Create a clock
    // Create a handler for animation
    EventHandler<ActionEvent> eventHandler = e -> {
      clock.setCurrentTime(); // Set a new clock time
    };
    // Create an animation for a running clock
    Timeline animation = new Timeline(
      new KeyFrame(Duration.millis(1000), eventHandler));
    animation.setCycleCount(Timeline.INDEFINITE);
    animation.play(); // Start animation
    Scene scene = new Scene(clock, 250, 250);
    primaryStage.setTitle("ClockAnimation");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

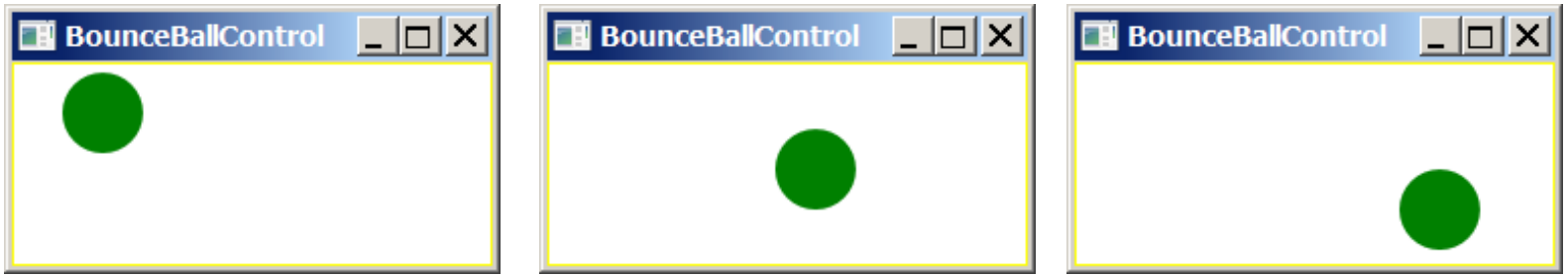


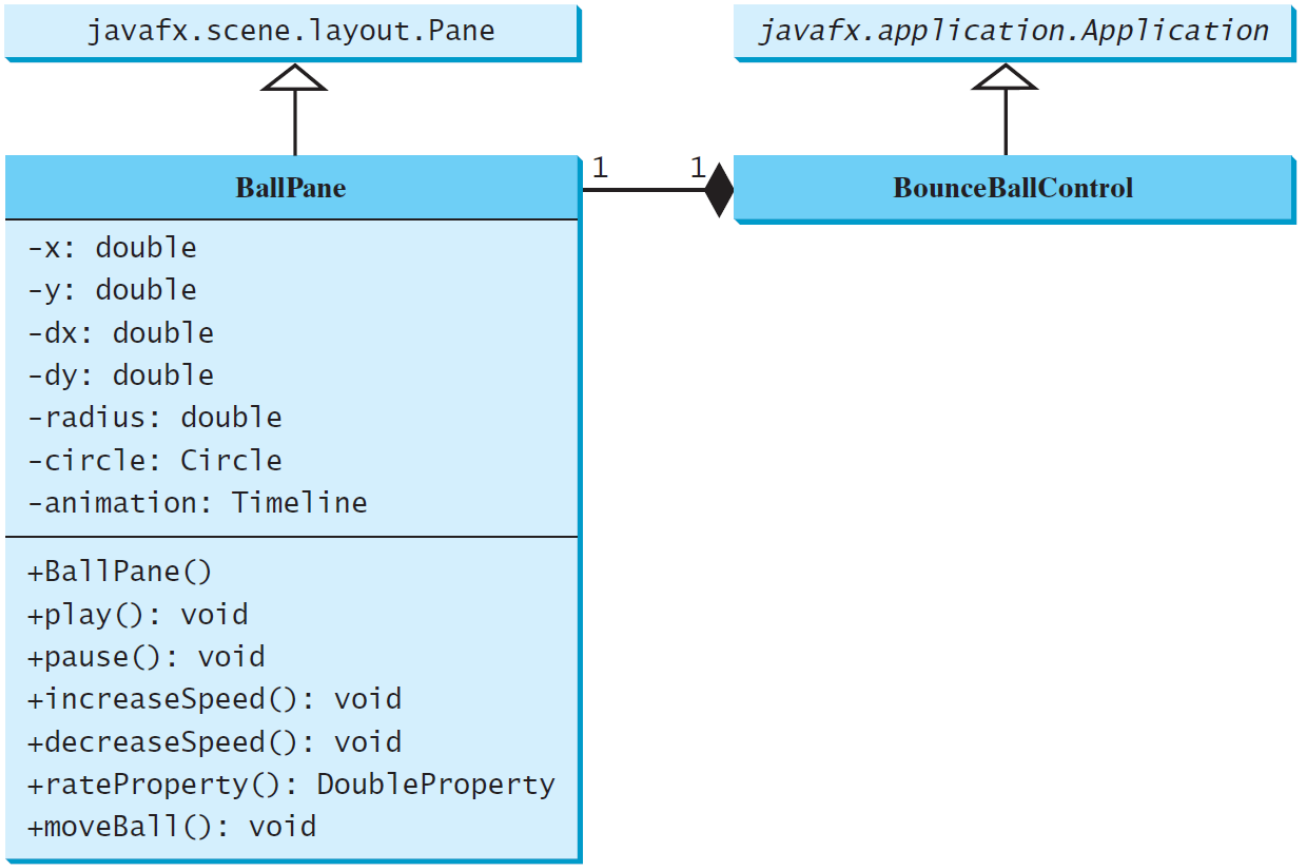
```
// ClockPane:
import java.util.Calendar;
import java.util.GregorianCalendar;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
public class ClockPane extends Pane {
    private int hour;
    private int minute;
    private int second;
    // Clock pane's width and height
    private double w = 250, h = 250;
    public ClockPane() {
        setCurrentTime();
    }
    public ClockPane(int hour, int minute, int second) {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
        paintClock();
    }
    public int getHour() {
        return hour;
    }
    public void setHour(int hour) {
        this.hour = hour;
        paintClock();
    }
}
```

```
public int getMinute() {
    return minute;
}
public void setMinute(int minute) {
    this.minute = minute;
    paintClock();
}
public int getSecond() {
    return second;
}
public void setSecond(int second) {
    this.second = second;
    paintClock();
}
public double getW() {
    return w;
}
public void setW(double w) {
    this.w = w;
    paintClock();
}
public double getH() {
    return h;
}
public void setH(double h) {
    this.h = h;
    paintClock();
}
public void setCurrentTime() {
    Calendar calendar = new GregorianCalendar();
    this.hour = calendar.get(Calendar.HOUR_OF_DAY);
    this.minute = calendar.get(Calendar.MINUTE);
    this.second = calendar.get(Calendar.SECOND);
    paintClock(); // Repaint the clock
}
```

```
private void paintClock() {
    // Initialize clock parameters
    double clockRadius = Math.min(w, h) * 0.8 * 0.5;
    double centerX = w / 2;
    double centerY = h / 2;
    // Draw circle
    Circle circle = new Circle(centerX, centerY, clockRadius);
    circle.setFill(Color.WHITE);
    circle.setStroke(Color.BLACK);
    Text t1 = new Text(centerX - 5, centerY - clockRadius + 12, "12");
    Text t2 = new Text(centerX - clockRadius + 3, centerY + 5, "9");
    Text t3 = new Text(centerX + clockRadius - 10, centerY + 3, "3");
    Text t4 = new Text(centerX - 3, centerY + clockRadius - 3, "6");
    // Draw second hand
    double sLength = clockRadius * 0.8;
    double secondX = centerX + sLength * Math.sin(second * (2 * Math.PI / 60));
    double secondY = centerY - sLength * Math.cos(second * (2 * Math.PI / 60));
    Line sLine = new Line(centerX, centerY, secondX, secondY);
    sLine.setStroke(Color.RED);
    // Draw minute hand
    double mLength = clockRadius * 0.65;
    double xMinute = centerX + mLength * Math.sin(minute * (2 * Math.PI / 60));
    double minuteY = centerY - mLength * Math.cos(minute * (2 * Math.PI / 60));
    Line mLine = new Line(centerX, centerY, xMinute, minuteY);
    mLine.setStroke(Color.BLUE);
    // Draw hour hand
    double hLength = clockRadius * 0.5;
    double hourX = centerX + hLength * Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    double hourY = centerY - hLength * Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    Line hLine = new Line(centerX, centerY, hourX, hourY);
    hLine.setStroke(Color.GREEN);
    getChildren().clear();
    getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine);
}
```

Bouncing Ball





```
import javafx.scene.layout.Pane;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.beans.property.DoubleProperty;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.util.Duration;
public class BallPane extends Pane {
    public final double radius = 20;
    private double x = radius, y = radius;
    private double dx = 1, dy = 1;
    private Circle circle = new Circle(x, y, radius);
    private Timeline animation;
    public BallPane() {
        circle.setFill(Color.GREEN); // Set ball color
        getChildren().add(circle); // Place a ball into this pane
        // Create an animation for moving the ball
        animation = new Timeline(new KeyFrame(Duration.millis(50), e -> moveBall()));
        animation.setCycleCount(Timeline.INDEFINITE);
        animation.play(); // Start animation
    }
    public void play() {
        animation.play();
    }
    public void pause() {
        animation.pause();
    }
    public void increaseSpeed() {
        animation.setRate(animation.getRate() + 0.1);
    }
    public void decreaseSpeed() {
        animation.setRate(
                animation.getRate() > 0 ? animation.getRate() - 0.1 : 0);
    }
```

```
    public DoubleProperty rateProperty() {
        return animation.rateProperty();
    }

    protected void moveBall() {
        // Check boundaries
        if (x < radius || x > getWidth() - radius) {
            dx *= -1; // Change ball move direction
        }
        if (y < radius || y > getHeight() - radius) {
            dy *= -1; // Change ball move direction
        }

        // Adjust ball position
        x += dx;
        y += dy;
        circle.setCenterX(x);
        circle.setCenterY(y);
    }
}
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.KeyCode;
public class BounceBallControl extends Application {
    @Override
    public void start(Stage primaryStage) {
        BallPane ballPane = new BallPane(); // Create a ball pane
        // Pause and resume animation
        ballPane.setOnMousePressed(e -> ballPane.pause());
        ballPane.setOnMouseReleased(e -> ballPane.play());
        // Increase and decrease animation
        ballPane.setOnKeyPressed(e -> {
            if (e.getCode() == KeyCode.UP) {
                ballPane.increaseSpeed();
            } else if (e.getCode() == KeyCode.DOWN) {
                ballPane.decreaseSpeed();
            }
        });
        Scene scene = new Scene(ballPane, 250, 150);
        primaryStage.setTitle("BounceBallControl");
        primaryStage.setScene(scene);
        primaryStage.show();
        // Must request focus after the primary stage is displayed
        ballPane.requestFocus();
    }

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

JavaFX support for mobile devices

- JavaFX has event programming support for mobile devices:

javafx.scene.input.SwipeEvent,
javafx.scene.input.TouchEvent,
javafx.scene.input.ZoomEvent.

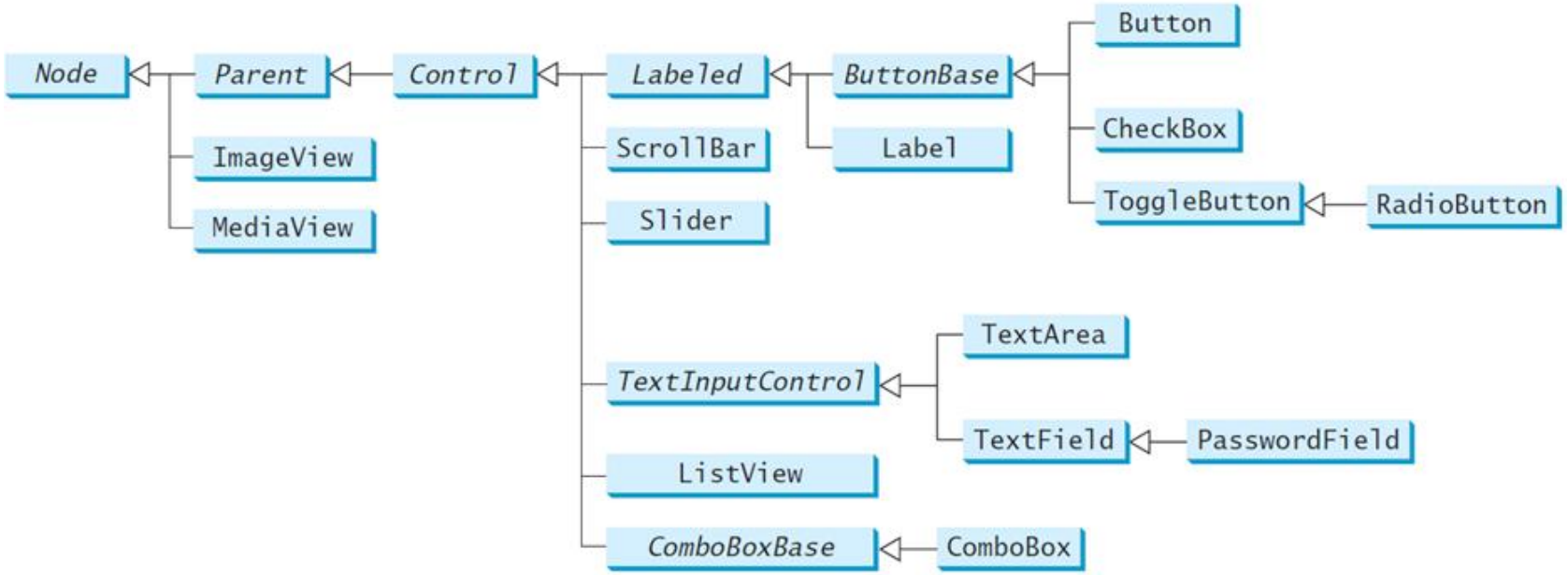
- Example:

http://docs.oracle.com/javase/8/javafx/events-tutorial/gestureeventsjava.htm

http://docs.oracle.com/javase/8/javafx/events-tutorial/toucheventsjava.htm

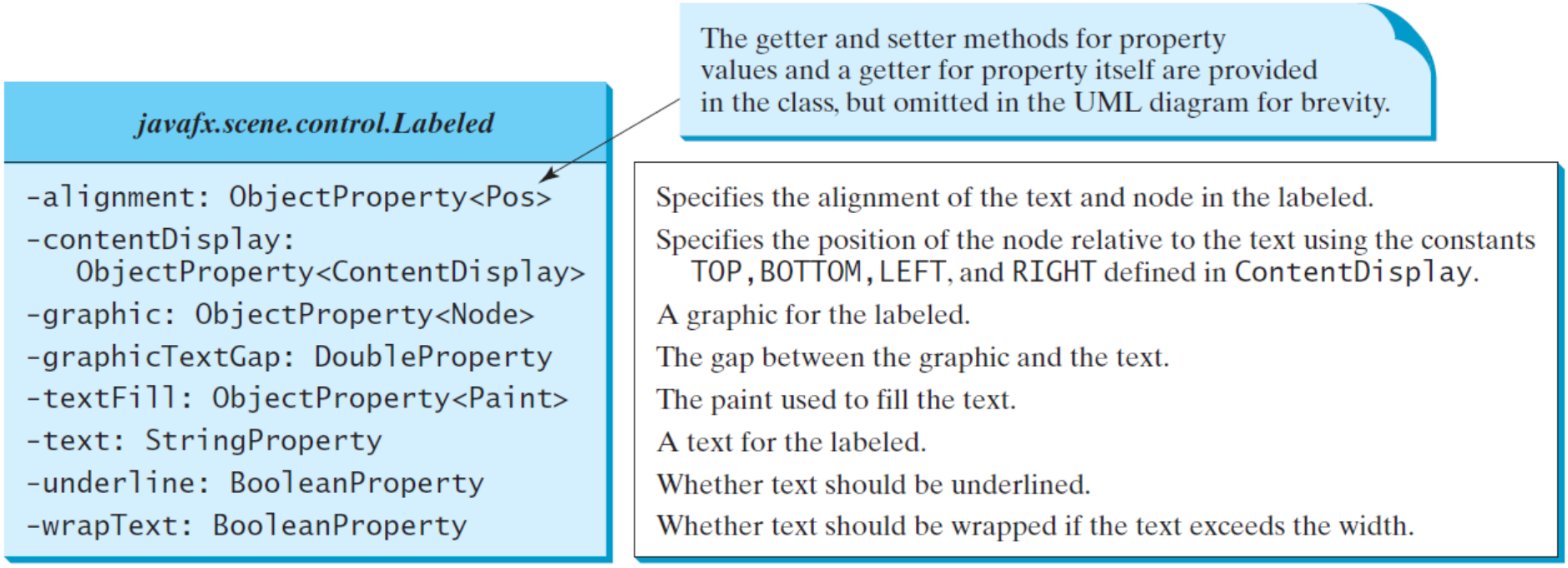
Control Nodes

- Input control nodes:

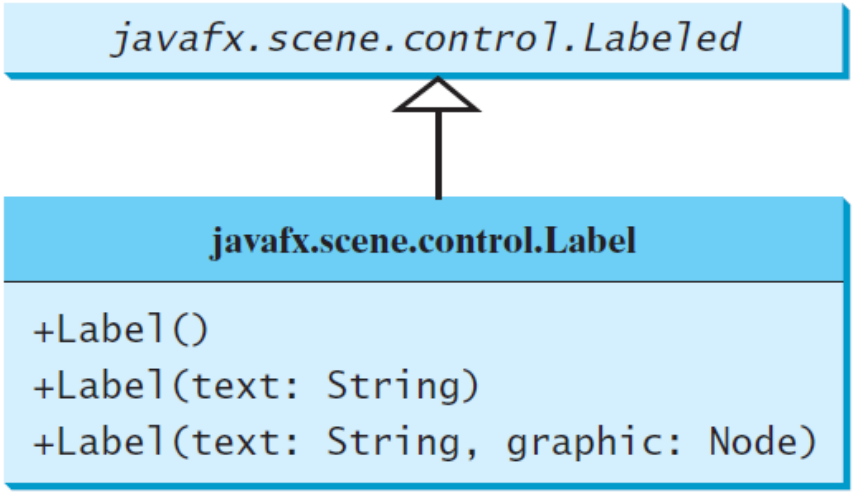


Labeled class

- 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



Label class



Creates an empty label.

Creates a label with the specified text.

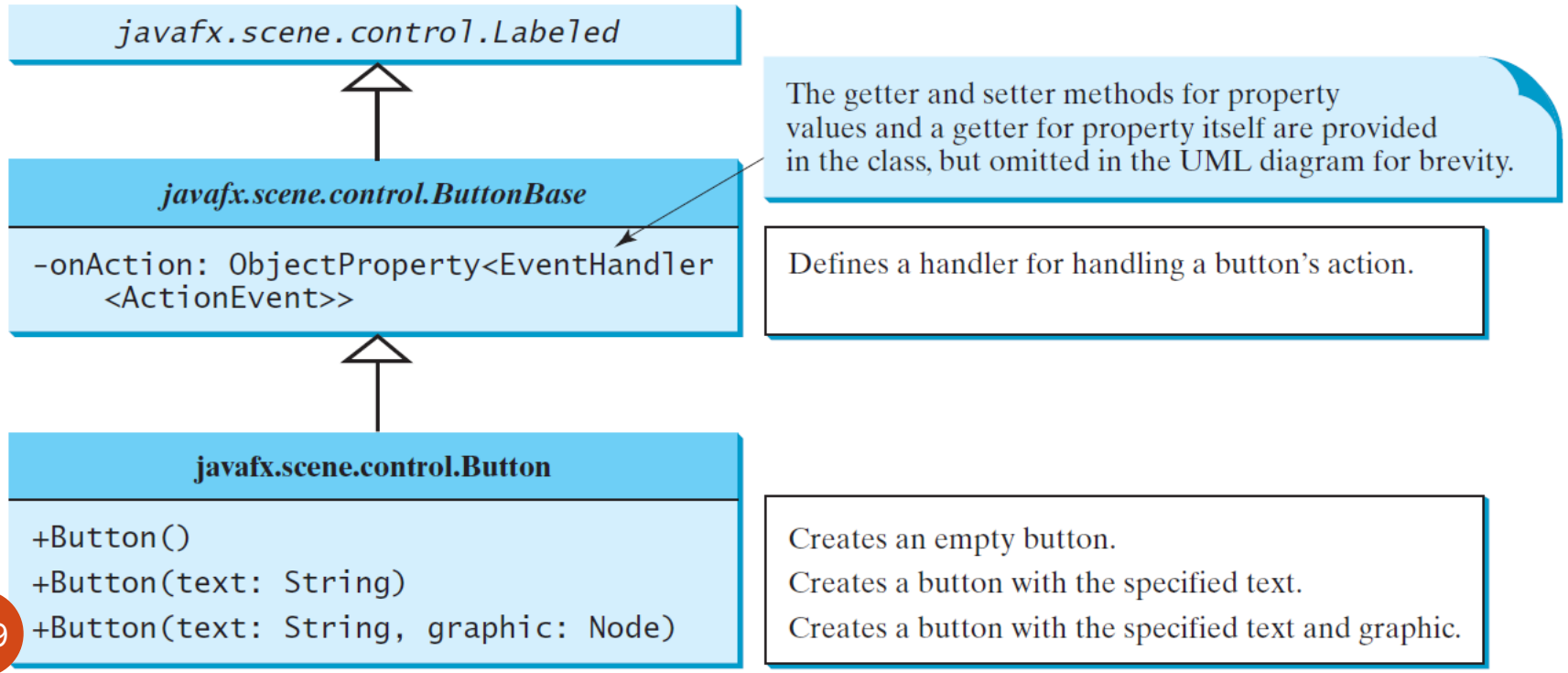
Creates a label with the specified text and graphic.

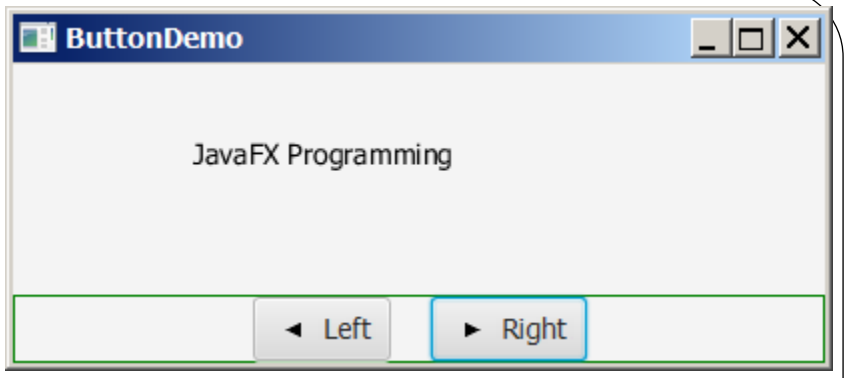
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
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
    public void start(Stage primaryStage) {
        ImageView us = new ImageView(new Image("us.jpg"));
        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);

    Scene scene = new Scene(pane, 700, 150);
    primaryStage.setTitle("LabelWithGraphic");
    primaryStage.setScene(scene);
    primaryStage.show();
}

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.



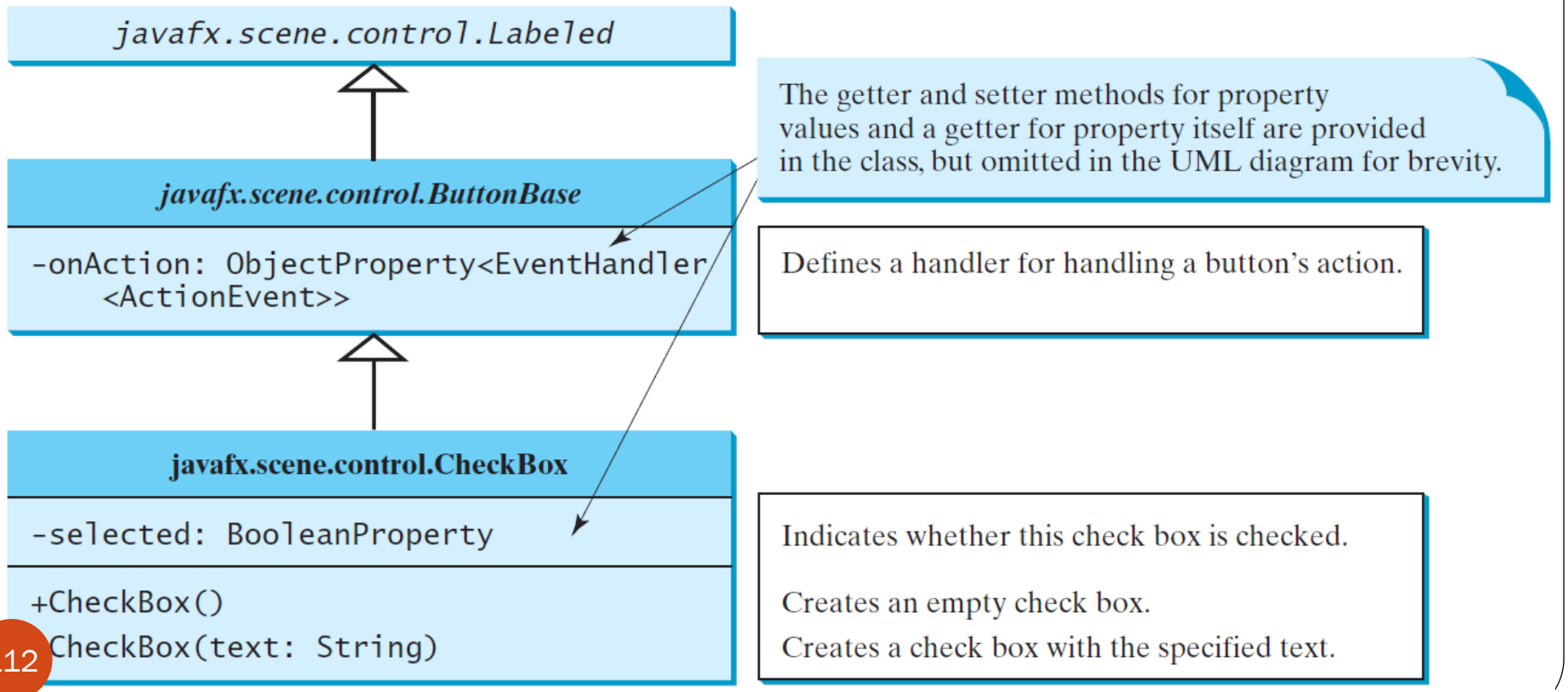


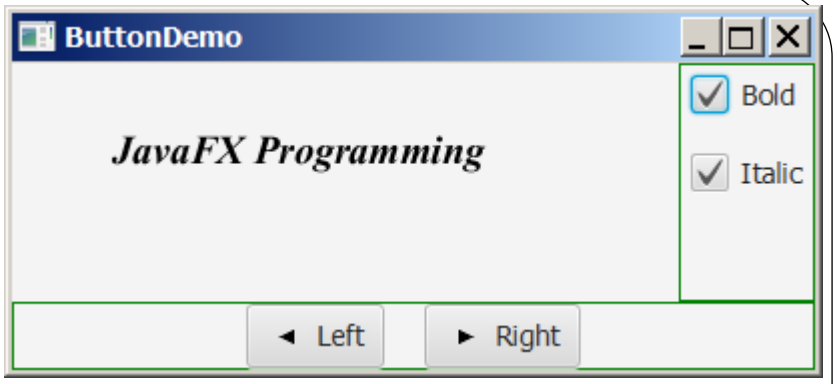

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;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

public class ButtonDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Scene scene = new Scene(getPane(), 450, 200);
        primaryStage.setTitle("ButtonDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    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;
    }

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

CheckBox

- A CheckBox is used for the user to make a selection (square box).
- CheckBox inherits all the properties from ButtonBase and Labeled: onAction, text, graphic, alignment, graphicTextGap, textFill, contentDisplay.



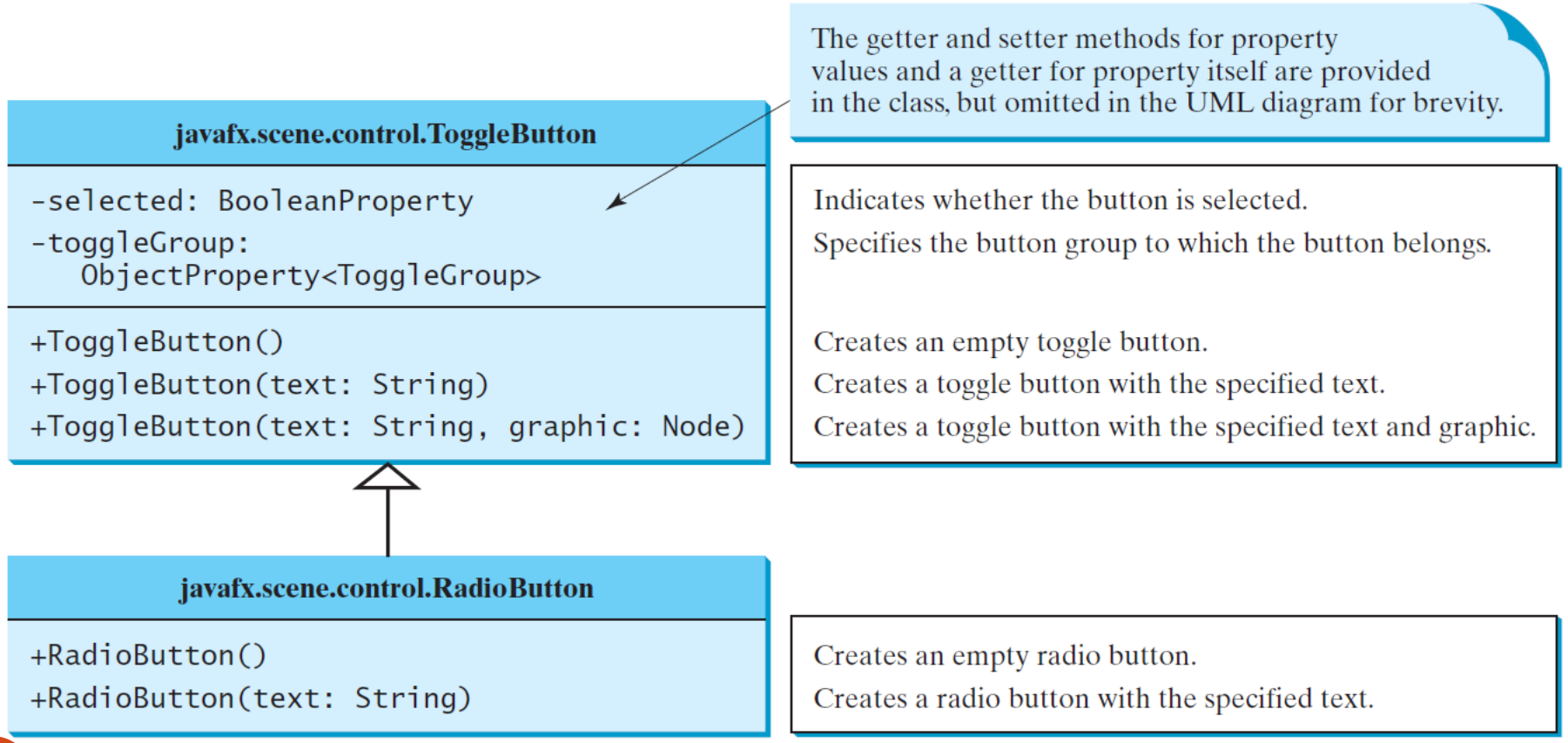



```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;

public class CheckBoxDemo extends ButtonDemo {
    @Override // Override the getPane() method in the super class
    protected BorderPane getPane() {
        BorderPane pane = super.getPane();

        Font fontBoldItalic = Font.font("Times New Roman",
                FontWeight.BOLD, FontPosture.ITALIC, 20);
        Font fontBold = Font.font("Times New Roman",
                FontWeight.BOLD, FontPosture.REGULAR, 20);
        Font fontItalic = Font.font("Times New Roman",
                FontWeight.NORMAL, FontPosture.ITALIC, 20);
        Font fontNormal = Font.font("Times New Roman",
                FontWeight.NORMAL, FontPosture.REGULAR, 20);

        text.setFont(fontNormal);

        VBox paneForCheckBoxes = new VBox(20);
        paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
        paneForCheckBoxes.setStyle("-fx-border-color: green");
```

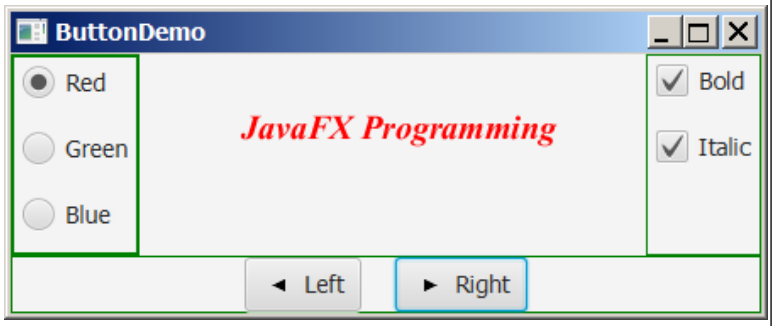
```
        CheckBox chkBold = new CheckBox("Bold");
        CheckBox chkItalic = new CheckBox("Italic");
        paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
        pane.setRight(paneForCheckBoxes);

        EventHandler<ActionEvent> handler = e -> {
            if (chkBold.isSelected() && chkItalic.isSelected()) {
                text.setFont(fontBoldItalic); // Both check boxes checked
            } else if (chkBold.isSelected()) {
                text.setFont(fontBold); // The Bold check box checked
            } else if (chkItalic.isSelected()) {
                text.setFont(fontItalic); // The Italic check box checked
            } else {
                text.setFont(fontNormal); // Both check boxes unchecked
            }
        };

        chkBold.setOnAction(handler);
        chkItalic.setOnAction(handler);

        return pane; // Return a new pane
    }

    // the start method is inherited from the superclass ButtonDemo

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

RadioButton

- Radio buttons allow to choose a **single** item from a group of choices.
  - Radio buttons display a circle that is either filled (if selected) or blank (if not selected).



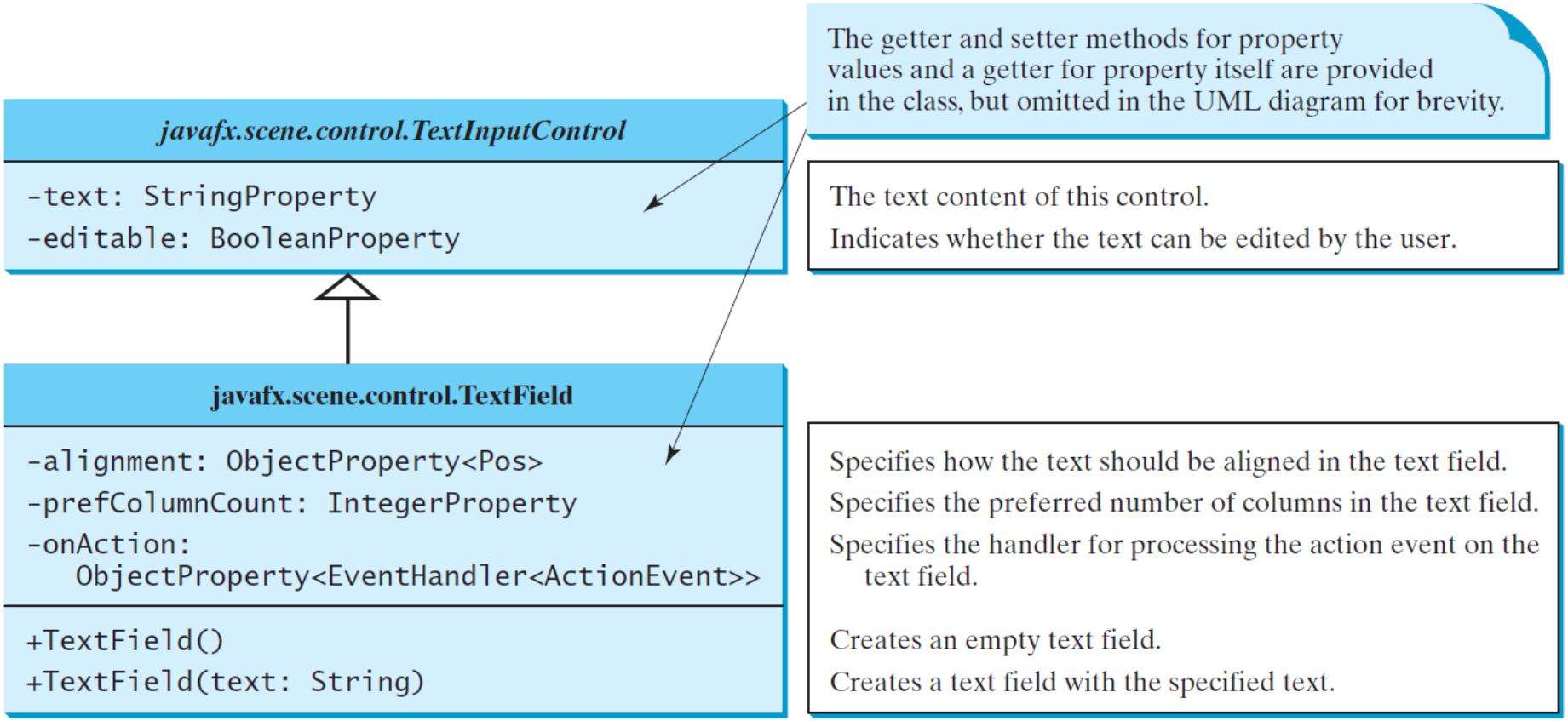
```
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

public class RadioButtonDemo extends CheckBoxDemo {
    @Override // Override the getPane() method in the super class
    protected BorderPane getPane() {
        BorderPane pane = super.getPane();
        VBox paneForRadioButtons = new VBox(20);
        paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));
        paneForRadioButtons.setStyle("-fx-border-color: green");
        RadioButton rbRed = new RadioButton("Red");
        RadioButton rbGreen = new RadioButton("Green");
        RadioButton rbBlue = new RadioButton("Blue");
        paneForRadioButtons.getChildren().addAll(rbRed, rbGreen, rbBlue);
        pane.setLeft(paneForRadioButtons);
        ToggleGroup group = new ToggleGroup();
        rbRed.setToggleGroup(group);
        rbGreen.setToggleGroup(group);
        rbBlue.setToggleGroup(group);

        rbRed.setOnAction(e -> {
            if (rbRed.isSelected()) {
                text.setFill(Color.RED);
            }
        });
```



```
        rbGreen.setOnAction(e -> {
            if (rbGreen.isSelected()) {
                text.setFill(Color.GREEN);
            }
        });

        rbBlue.setOnAction(e -> {
            if (rbBlue.isSelected()) {
                text.setFill(Color.BLUE);
            }
        });

        return pane;
    }

    // the start method is inherited from the superclass ButtonDemo

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

TextField

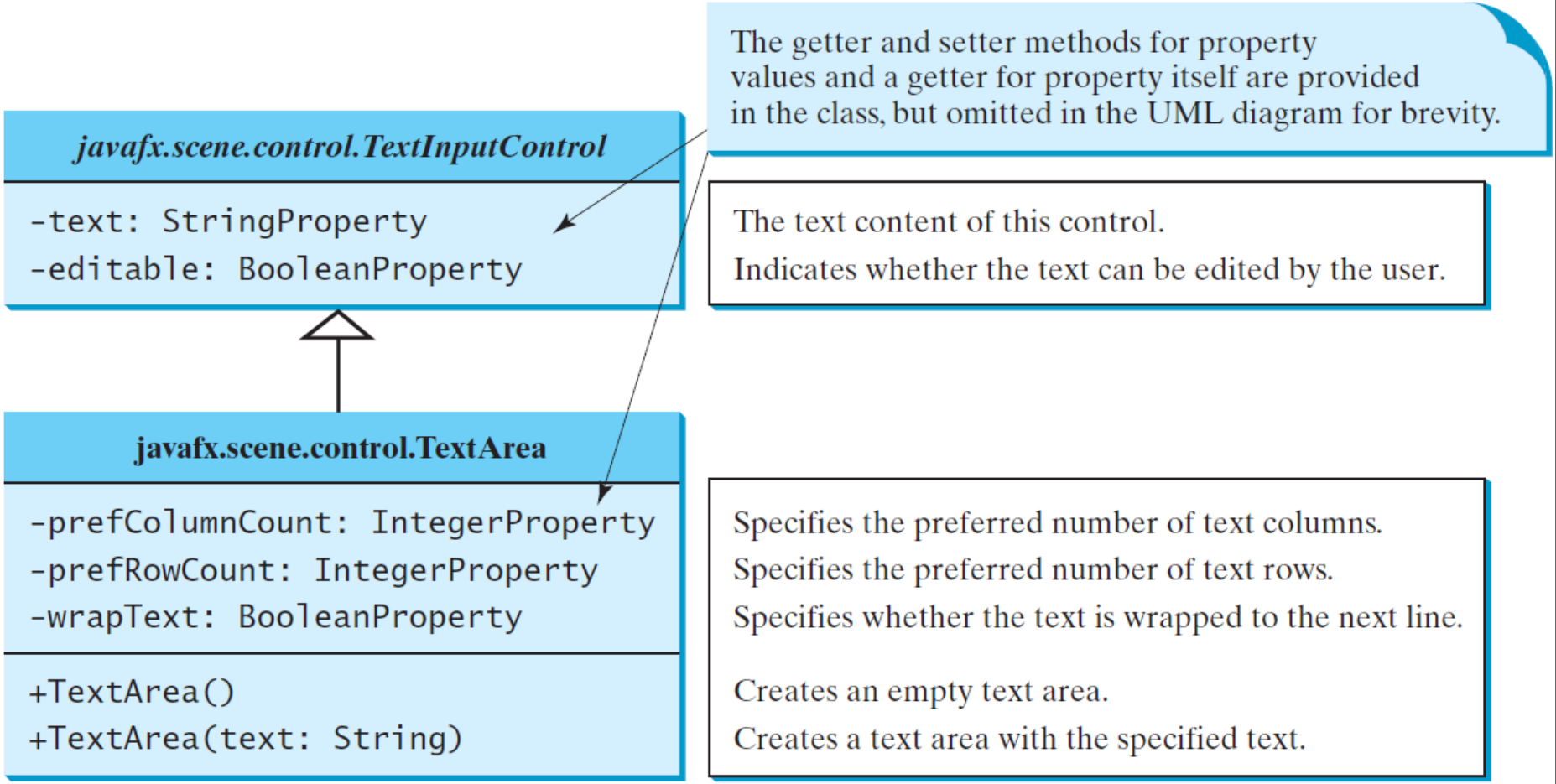
- A text field can be used to enter or display a string. TextField is a subclass of TextInputControl.



```
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
public class TextFieldDemo extends RadioButtonDemo{
    @Override
    protected BorderPane getPane() {
        BorderPane pane = super.getPane();

        BorderPane paneForTextField = new BorderPane();
        paneForTextField.setPadding(new Insets(5, 5, 5, 5));
        paneForTextField.setStyle("-fx-border-color: green");
        paneForTextField.setLeft(new Label("Enter a new message: "));

        TextField tf = new TextField();
        tf.setAlignment(Pos.BOTTOM_RIGHT);
        paneForTextField.setCenter(tf);
        pane.setTop(paneForTextField);

        tf.setOnAction(e -> text.setText(tf.getText()));

        return pane;
    }

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

TextArea

- A TextArea enables the user to enter multiple lines of text.



ComboBox

- A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.

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.ComboBoxBase<T>

```
-value: ObjectProperty<T>
-editable: BooleanProperty
-onAction:
   ObjectProperty<EventHandler<ActionEvent>>
```

The value selected in the combo box.

Specifies whether the combo box allows user input.

Specifies the handler for processing the action event.

javafx.scene.control.ComboBox<T>

```
-items: ObjectProperty<ObservableList<T>>
-visibleRowCount: IntegerProperty

+ComboBox()
+ComboBox(items: ObservableList<T>)
```

The items in the combo box popup.

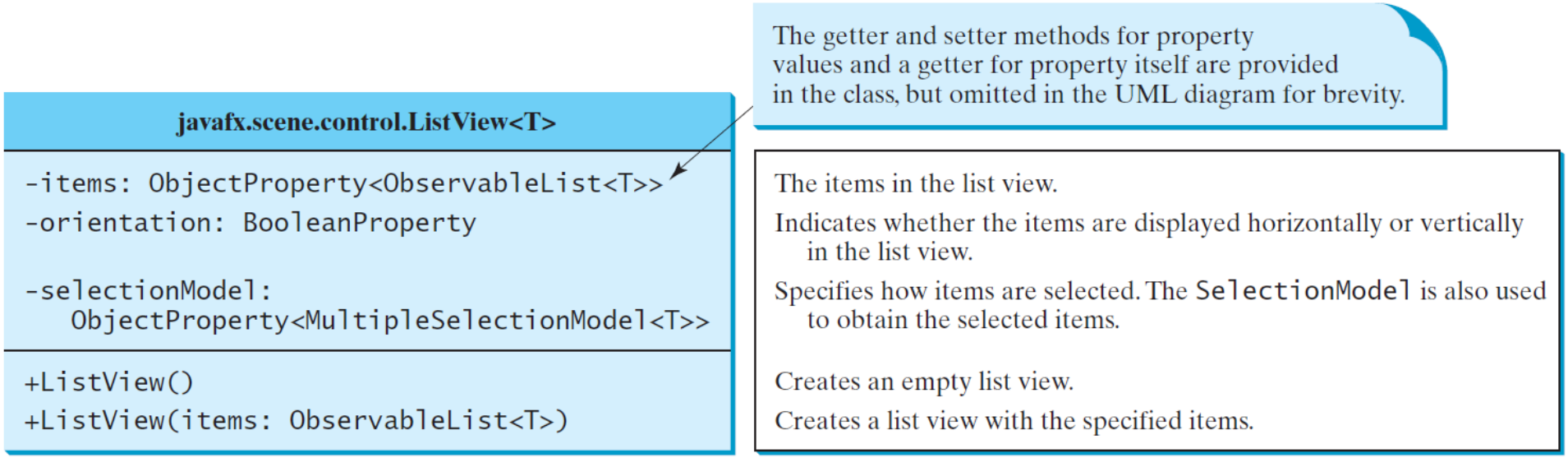
The maximum number of visible rows of the items in the combo box popup.

Creates an empty combo box.

Creates a combo box with the specified items.

ListView

- A list view is a component that performs basically the same function as a combo box, but it enables the user to choose a single value or multiple values.

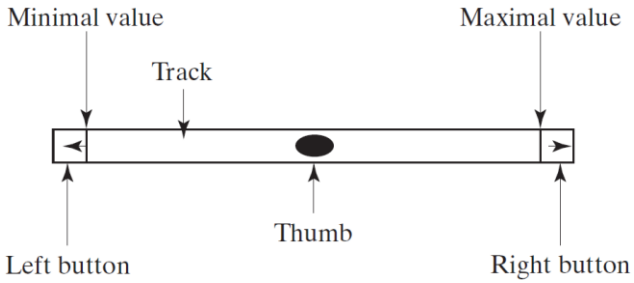


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.SelectionMode;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.collections.FXCollections;

public class ListViewDemo extends Application {
  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"United States of America", "Canada", "China",
    "Denmark", "France", "Germany", "India"};
  // Declare an ImageView array for the national flags
  private ImageView[] ImageViews = {
    new ImageView("image/us.gif"),
    new ImageView("image/ca.gif"),
    new ImageView("image/china.gif"),
    new ImageView("image/denmark.gif"),
    new ImageView("image/fr.gif"),
    new ImageView("image/germany.gif"),
    new ImageView("image/india.gif")
  };
  @Override
  public void start(Stage primaryStage) {
    ListView<String> lv = new ListView<>(FXCollections
                                        .observableArrayList(flagTitles));
    lv.setPrefSize(400, 400);
```

```
    lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

    // Create a pane to hold image views
    FlowPane imagePane = new FlowPane(10, 10);
    BorderPane pane = new BorderPane();
    pane.setLeft(new ScrollPane(lv));
    pane.setCenter(imagePane);

    lv.getSelectionModel().selectedItemProperty().addListener(
      ov -> {
        imagePane.getChildren().clear();
        for (Integer i: lv.getSelectionModel().getSelectedIndices()) {
          imagePane.getChildren().add(ImageViews[i]);
        }
    });

    Scene scene = new Scene(pane, 450, 170);
    primaryStage.setTitle("ListViewDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }


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

ScrollBar

- A scroll bar is a control that enables the user to select from a range of values. The scrollbar appears in two styles: horizontal and vertical.



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.ScrollBar

```
-blockIncrement: DoubleProperty
-max: DoubleProperty
-min: DoubleProperty
-unitIncrement: DoubleProperty

-value: DoubleProperty
-visibleAmount: DoubleProperty
-orientation: ObjectProperty<Orientation>

+ScrollBar()
+increment()
+decrement()
```

The amount to adjust the scroll bar if the track of the bar is clicked (default: 10).

The maximum value represented by this scroll bar (default: 100).

The minimum value represented by this scroll bar (default: 0).

The amount to adjust the scroll bar when the `increment()` and `decrement()` methods are called (default: 1).

Current value of the scroll bar (default: 0).

The width of the scroll bar (default: 15).

Specifies the orientation of the scroll bar (default: `HORIZONTAL`).

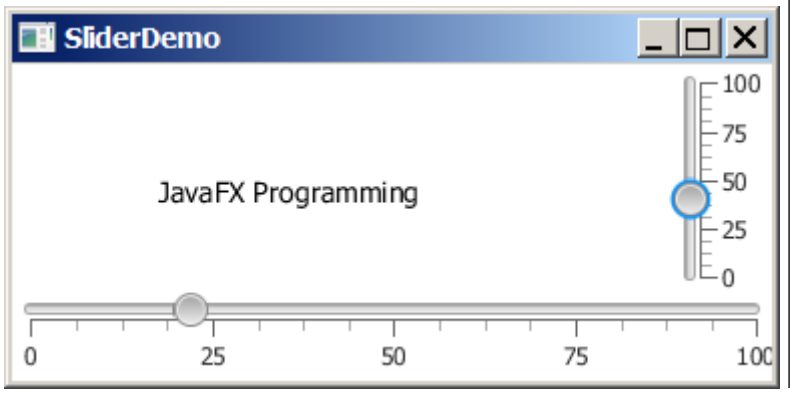
Creates a default horizontal scroll bar.

Increments the value of the scroll bar by `unitIncrement`.

Decrements the value of the scroll bar by `unitIncrement`.

Slider

- Slider is similar to ScrollBar, but Slider has more properties and can appear in many forms.



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.Slider

```
-blockIncrement: DoubleProperty
-max: DoubleProperty
-min: DoubleProperty
-value: DoubleProperty
-orientation: ObjectProperty<Orientation>
-majorTickUnit: DoubleProperty
-minorTickCount: IntegerProperty
-showTickLabels: BooleanProperty
-showTickMarks: BooleanProperty
```

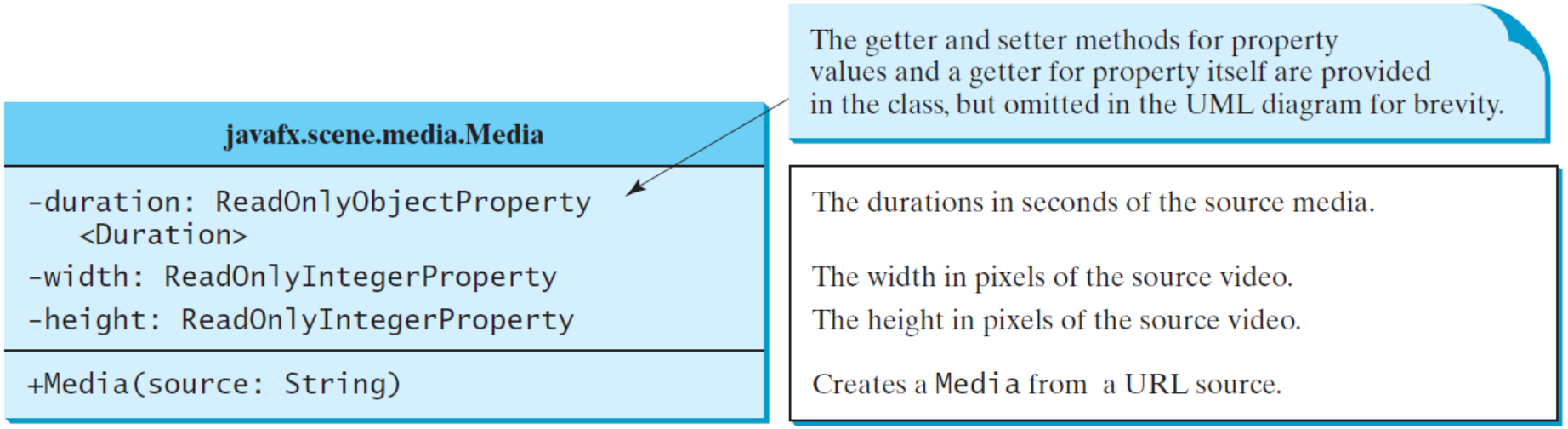
```
+Slider()
+Slider(min: double, max: double,
   value: double)
```

The amount to adjust the slider if the track of the bar is clicked (default: 10).
The maximum value represented by this slider (default: 100).
The minimum value represented by this slider (default: 0).
Current value of the slider (default: 0).
Specifies the orientation of the slider (default: HORIZONTAL).
The unit distance between major tick marks.
The number of minor ticks to place between two major ticks.
Specifies whether the labels for tick marks are shown.
Specifies whether the tick marks are shown.

Creates a default horizontal slider.
Creates a slider with the specified min, max, and value.

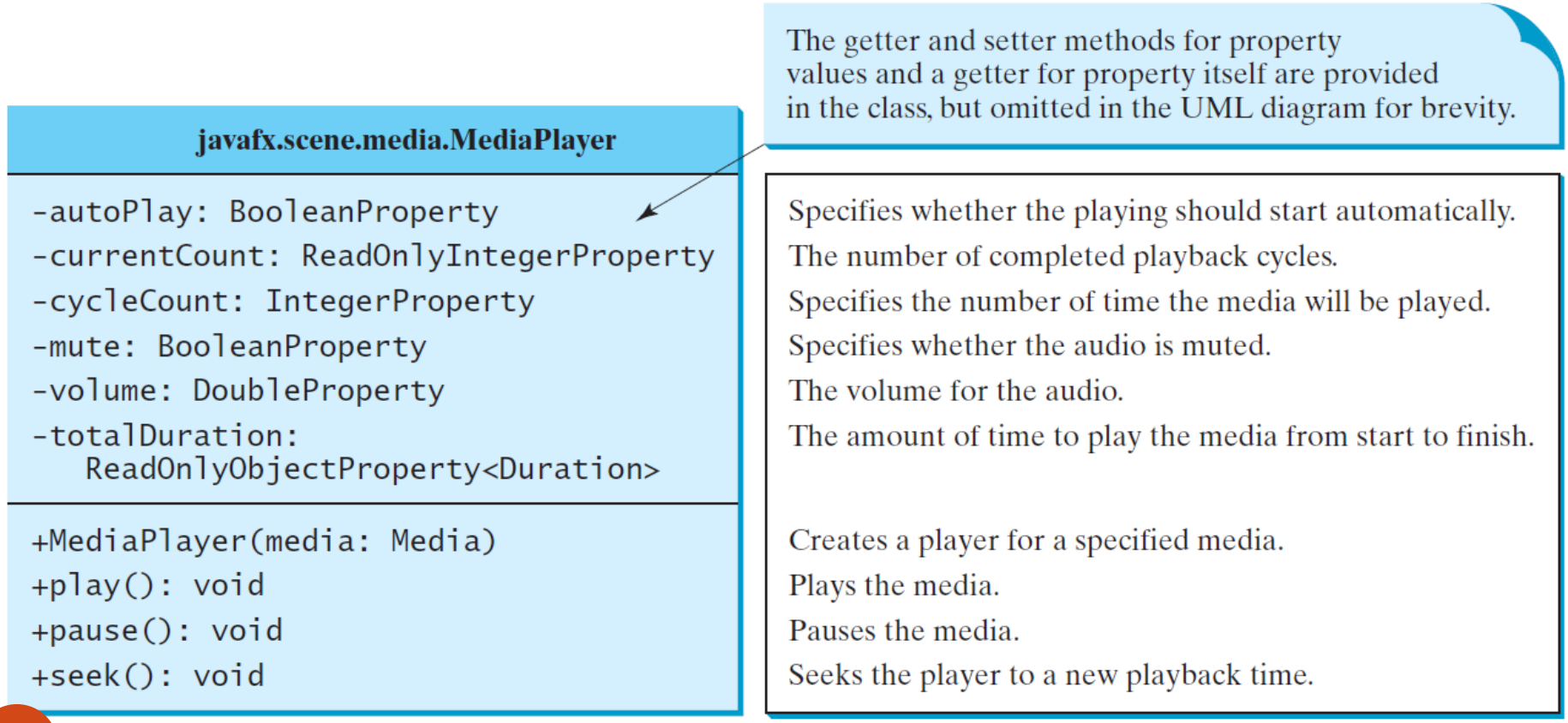
Media

- The Media class is used to obtain the source of a media type.
- The MediaPlayer class is used to play and control the media.
- The MediaView class is used to display video.



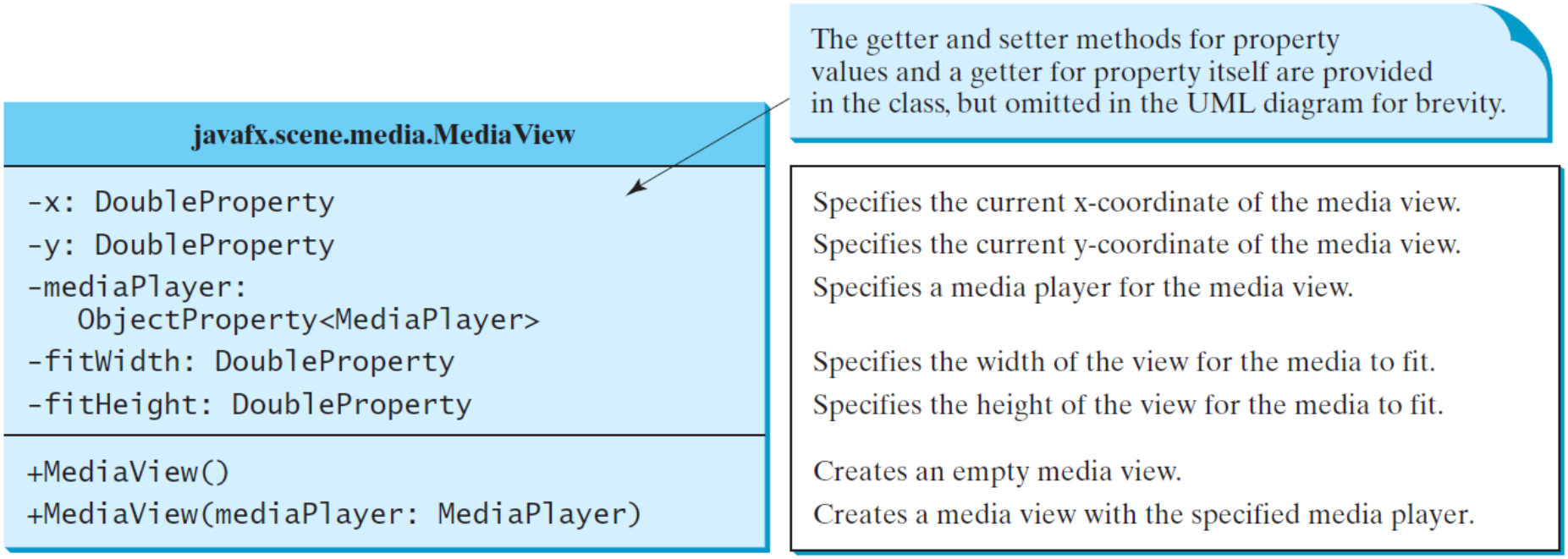
MediaPlayer

- The MediaPlayer class plays and controls media with properties: autoPlay, currentCount, cycleCount, mute, volume, and totalDuration.



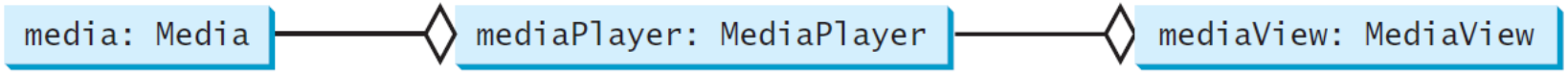
MediaView

- The MediaView class is a subclass of Node that provides a view of the Media being played by a MediaPlayer.
  - The MediaView class provides the properties for viewing the media.



Example: Using Media

- This example displays a video in a view: use the play/pause button to play or pause the video and use the rewind button to restart the video, and use the slider to control the volume of the audio.

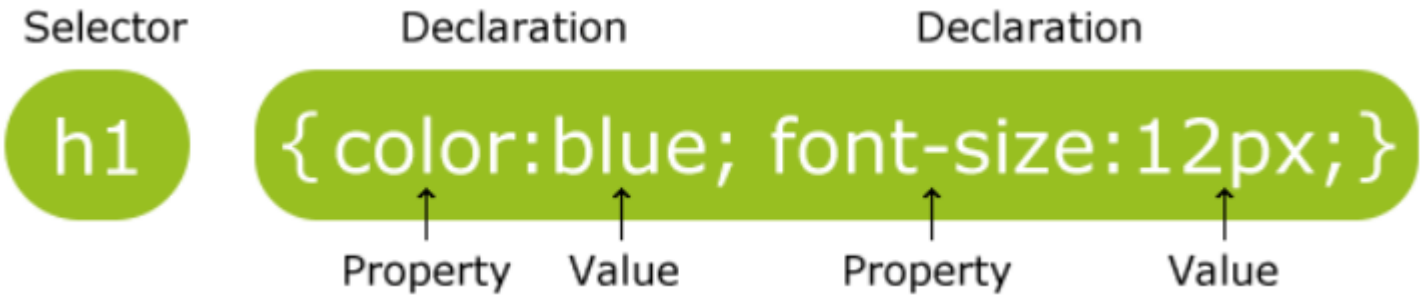


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Region;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.geometry.Pos;
import javafx.util.Duration;
public class MediaDemo extends Application {
    private static final String MEDIA_URL = "sample.mp4";
    @Override
    public void start(Stage primaryStage) {
        Media media = new Media(MEDIA_URL);
        MediaPlayer mediaPlayer = new MediaPlayer(media);
        MediaView mediaView = new MediaView(mediaPlayer);
        Button playButton = new Button(">");
        playButton.setOnAction(e -> {
            if (playButton.getText().equals(">")) {
                mediaPlayer.play();
                playButton.setText("||");
            } else {
                mediaPlayer.pause();
                playButton.setText(">");
            }
        });
```

```
        Button rewindButton = new Button("<<");
        rewindButton.setOnAction(e -> mediaPlayer.seek(Duration.ZERO));
        Slider slVolume = new Slider();
        slVolume.setPrefWidth(150);
        slVolume.setMaxWidth(Region.USE_PREF_SIZE);
        slVolume.setMinWidth(30);
        slVolume.setValue(50);
        mediaPlayer.volumeProperty().bind(slVolume.valueProperty().divide(100));
        HBox hBox = new HBox(10);
        hBox.setAlignment(Pos.CENTER);
        hBox.getChildren().addAll(playButton, rewindButton,
                new Label("Volume"), slVolume);
        BorderPane pane = new BorderPane();
        pane.setCenter(mediaView);
        pane.setBottom(hBox);

        Scene scene = new Scene(pane, 650, 500);
        primaryStage.setTitle("MediaDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

CSS

- Cascading Style Sheets (CSS) is a language used for describing the look and formatting of a document.
  - CSS is designed primarily to enable the separation of document content from document presentation (layout, colors, and fonts).
    - It is used to style web pages and user interfaces written in HTML, XHTML, and any kind of XML document.
  - The CSS language specifications are Web standards maintained by the World Wide Web Consortium (W3C).
  - CSS rule set example:



JavaFX CSS

- JavaFX Cascading Style Sheets (CSS) is based on the W3C CSS and allows to customize and develop themes for JavaFX controls and scene graph objects
  - http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
  - JavaFX uses the prefix "-fx-" to define its vendor CSS properties (separate from W3C CSS).
- A style sheet uses the style class or style id to define styles.
  - Mutiple style classes can be applied to a single node and a style id to a unique node.
  - The syntax .styleclass defines a style class.
  - The syntax #styleid defines a style id.

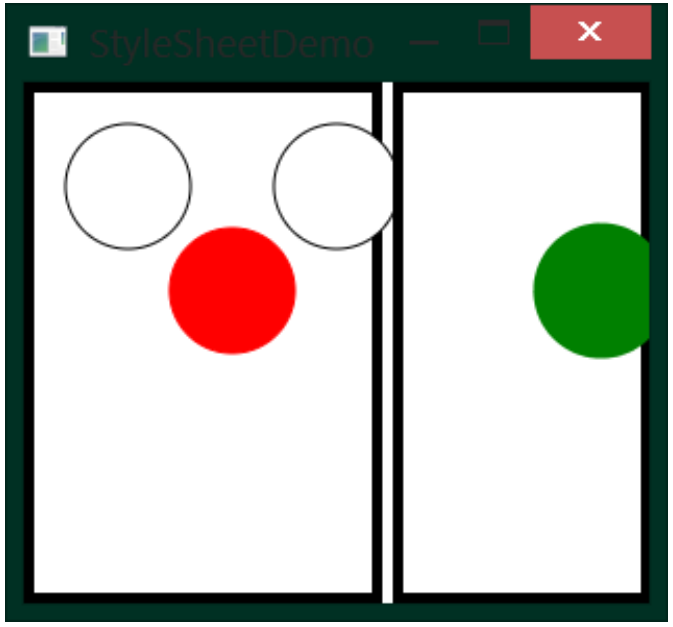
Style Class and Style ID

- **mystyle.css:**

```
.plaincircle {
  -fx-fill: white;
  -fx-stroke: black;
}
.circleborder {
  -fx-stroke-width: 5;
  -fx-stroke-dash-array: 12 2 4 2;
}
.border {
  -fx-border-color: black;
  -fx-border-width: 5;
}
#redcircle {
  -fx-fill: red;
  -fx-stroke: red;
}
#greencircle {
  -fx-fill: green;
  -fx-stroke: green;
}
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;
public class StyleSheetDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    HBox hBox = new HBox(5);
    Scene scene = new Scene(hBox, 300, 250);
    // Load the stylesheet
    scene.getStylesheets().add("mystyle.css");
    Pane pane1 = new Pane();
    Circle circle1 = new Circle(50, 50, 30);
    Circle circle2 = new Circle(150, 50, 30);
    Circle circle3 = new Circle(100, 100, 30);
    pane1.getChildren().addAll(circle1, circle2, circle3);
    pane1.getStyleClass().add("border");
    circle1.getStyleClass().add("plaincircle"); // Add a style class
    circle2.getStyleClass().add("plaincircle"); // Add a style class
    circle3.setId("redcircle"); // Add a style id
    Pane pane2 = new Pane();
    Circle circle4 = new Circle(100, 100, 30);
```

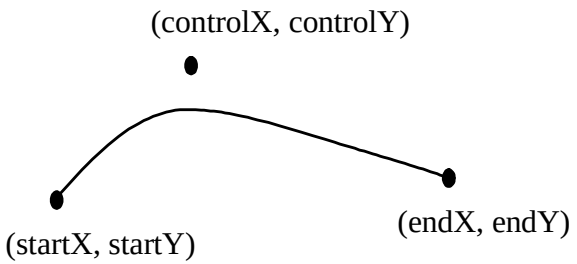
```
    circle4.getStyleClass().addAll("circleborder", "plainCircle");
    circle4.setId("greencircle"); // Add a style class
    pane2.getChildren().add(circle4);
    pane2.getStyleClass().add("border");

    hBox.getChildren().addAll(pane1, pane2);

    primaryStage.setTitle("StyleSheetDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

  // Lauch the program from command-line
  public static void main(String[] args) {
    launch(args);
  }
}
```



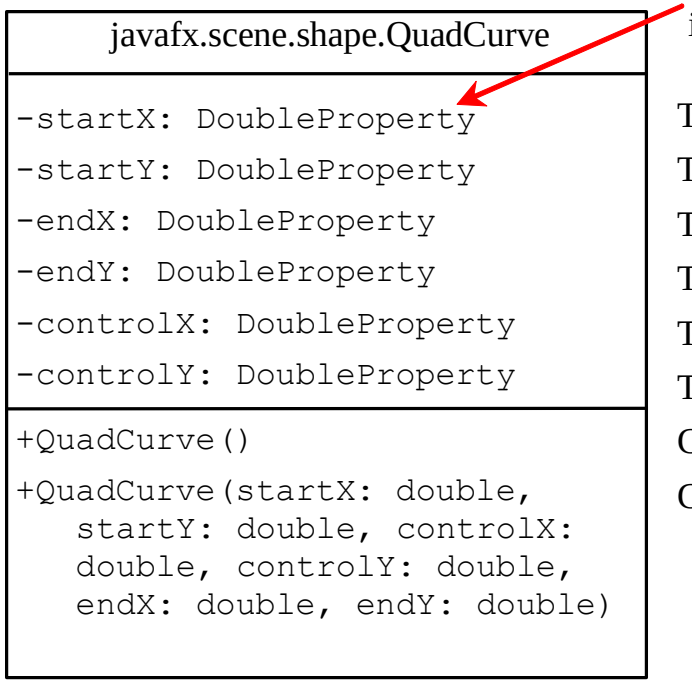
QuadCurve

- A quadratic curve is mathematically defined as a quadratic polynomial.

```
QuadCurve(double startX, double startY,
double controlX, double controlY, double
endX, double endY)
```



QuadCurve



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 x-coordinate of the start point (default 0).

The y-coordinate of the start point (default 0)..

The x-coordinate of the end point (default 0)..

The y-coordinate of the end point (default 0)..

The x-coordinate of the control point (default 0)..

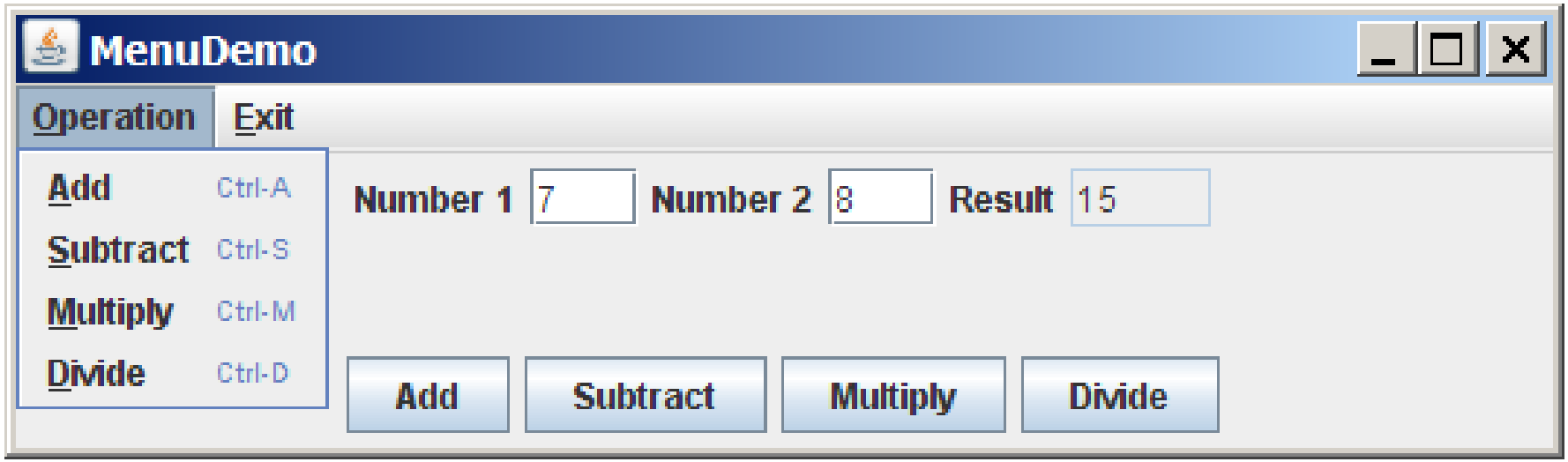
The y-coordinate of the control point (default 0)..

Creates an empty quad curve.

Creates a quad curve with the specified arguments.

Menus

- Menus make selection easier and are widely used in window applications.
  - JavaFX provides five classes that implement menus: MenuBar, Menu, MenuItem, CheckMenuItem, and RadioButtonMenuItem.
- MenuBar is a top-level menu component used to hold the menus.
  - A menu consists of menu items that the user can select (or toggle on or off).
  - A menu item can be an instance of MenuItem, CheckMenuItem, or RadioButtonMenuItem.
  - Menu items can be associated with nodes and keyboard accelerators.




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCombination;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.geometry.Pos;

public class MenuDemo extends Application {
  private TextField tfNumber1 = new TextField();
  private TextField tfNumber2 = new TextField();
  private TextField tfResult = new TextField();
```

```
@Override
 public void start(Stage primaryStage) {
   MenuBar menuBar = new MenuBar();

   Menu menuOperation = new Menu("Operation");
   Menu menuExit = new Menu("Exit");
   menuBar.getMenus().addAll(menuOperation, menuExit);

   MenuItem menuItemAdd = new MenuItem("Add");
   MenuItem menuItemSubtract = new MenuItem("Subtract");
   MenuItem menuItemMultiply = new MenuItem("Multiply");
   MenuItem menuItemDivide = new MenuItem("Divide");
   menuOperation.getItems().addAll(menuItemAdd, menuItemSubtract,
     menuItemMultiply, menuItemDivide);

   MenuItem menuItemClose = new MenuItem("Close");
   menuExit.getItems().add(menuItemClose);

   menuItemAdd.setAccelerator(
     KeyCombination.keyCombination("Ctrl+A"));
   menuItemSubtract.setAccelerator(
     KeyCombination.keyCombination("Ctrl+S"));
   menuItemMultiply.setAccelerator(
     KeyCombination.keyCombination("Ctrl+M"));
   menuItemDivide.setAccelerator(
     KeyCombination.keyCombination("Ctrl+D"));
   HBox hBox1 = new HBox(5);
   tfNumber1.setPrefColumnCount(2);
   tfNumber2.setPrefColumnCount(2);
   tfResult.setPrefColumnCount(2);
```

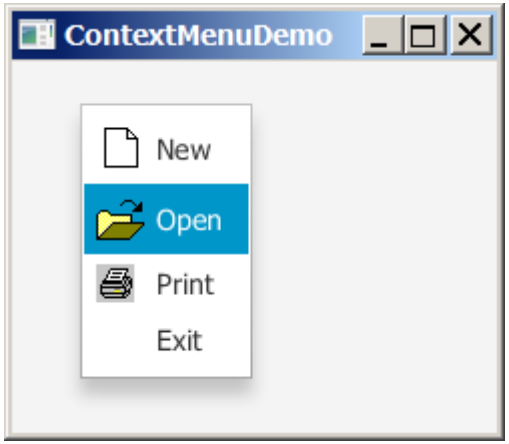
```
    hBox1.getChildren().addAll(new Label("Number 1:"), tfNumber1,
      new Label("Number 2:"), tfNumber2, new Label("Result:"),
      tfResult);
    hBox1.setAlignment(Pos.CENTER);

    HBox hBox2 = new HBox(5);
    Button btAdd = new Button("Add");
    Button btSubtract = new Button("Subtract");
    Button btMultiply = new Button("Multiply");
    Button btDivide = new Button("Divide");
    hBox2.getChildren().addAll(btAdd, btSubtract, btMultiply, btDivide);
    hBox2.setAlignment(Pos.CENTER);

    VBox vBox = new VBox(10);
    vBox.getChildren().addAll(menuBar, hBox1, hBox2);
    Scene scene = new Scene(vBox, 300, 250);
    primaryStage.setTitle("MenuDemo"); // Set the window title
    primaryStage.setScene(scene); // Place the scene in the window
    primaryStage.show(); // Display the window
    // Handle menu actions
    menuItemAdd.setOnAction(e -> perform('+'));
    menuItemSubtract.setOnAction(e -> perform('-'));
    menuItemMultiply.setOnAction(e -> perform('*'));
    menuItemDivide.setOnAction(e -> perform('/'));
    menuItemClose.setOnAction(e -> System.exit(0));
    // Handle button actions
    btAdd.setOnAction(e -> perform('+'));
    btSubtract.setOnAction(e -> perform('-'));
    btMultiply.setOnAction(e -> perform('*'));
    btDivide.setOnAction(e -> perform('/'));
  }
```

```
  private void perform(char operator) {
    double number1 = Double.parseDouble(tfNumber1.getText());
    double number2 = Double.parseDouble(tfNumber2.getText());

    double result = 0;
    switch (operator) {
      case '+': result = number1 + number2; break;
      case '-': result = number1 - number2; break;
      case '*': result = number1 * number2; break;
      case '/': result = number1 / number2; break;
    }

    tfResult.setText(result + "");
  };

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

Context Menu

- A context menu (also known as a popup menu) is like a regular menu, but does not have a menu bar and can float anywhere on the screen.
  - Creating a context menu is similar to creating a regular menu.
    - First, create an instance of ContextMenu, then add MenuItem, CheckMenuItem, and RadioMenuItem to the context menu.

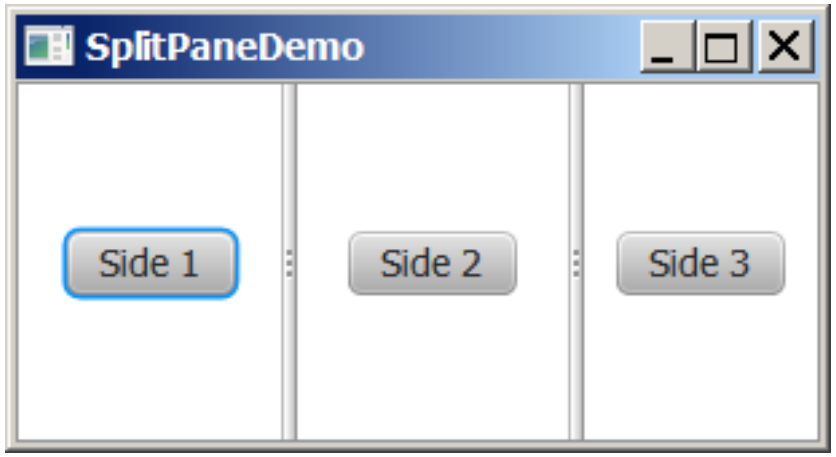
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.image.ImageView;
public class ContextMenuDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    ContextMenu contextMenu = new ContextMenu();
    MenuItem menuItemNew = new MenuItem("New",
      new ImageView("image/new.gif"));
    MenuItem menuItemOpen = new MenuItem("Open",
      new ImageView("image/open.gif"));
    MenuItem menuItemPrint = new MenuItem("Print",
      new ImageView("image/print.gif"));
    MenuItem menuItemExit = new MenuItem("Exit");
    contextMenu.getItems().addAll(menuItemNew, menuItemOpen,
      menuItemPrint, menuItemExit);
    Pane pane = new Pane();
    Scene scene = new Scene(pane, 300, 250);
    primaryStage.setTitle("ContextMenuDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
```

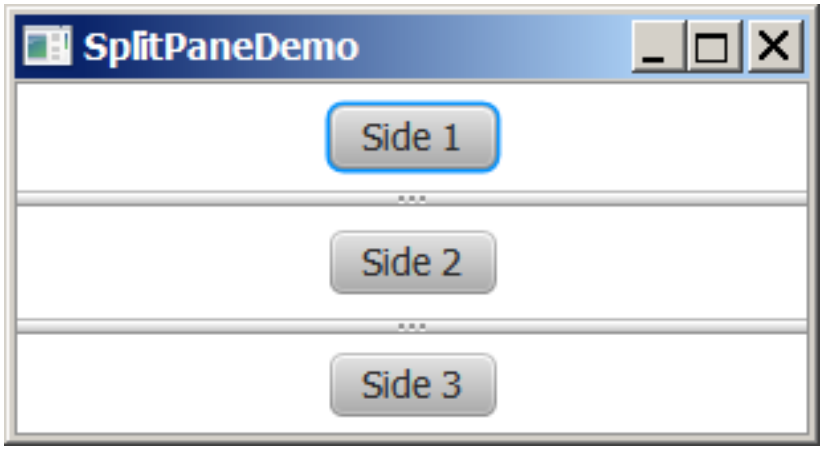


```
    pane.setOnMousePressed(
      e -> contextMenu.show(pane, e.getScreenX(), e.getScreenY()));

    menuItemNew.setOnAction(e -> System.out.println("New"));
    menuItemOpen.setOnAction(e -> System.out.println("Open"));
    menuItemPrint.setOnAction(e -> System.out.println("Print"));
    menuItemExit.setOnAction(e -> System.exit(0));
  }

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

SplitPane

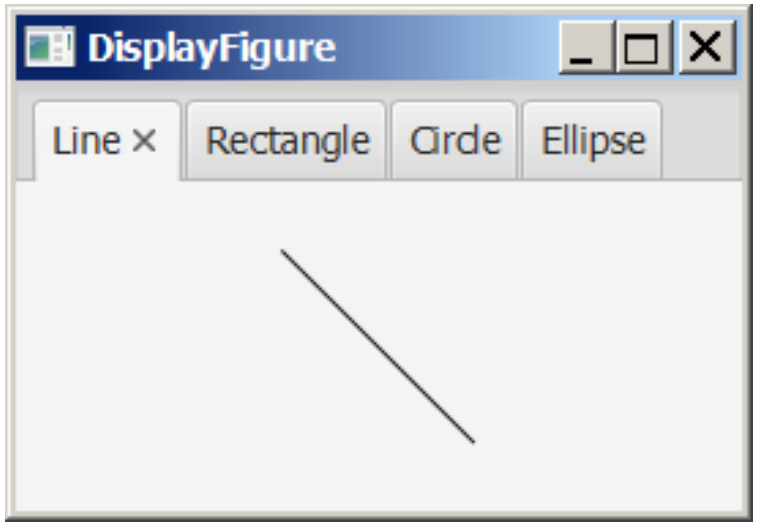
- The SplitPane class can be used to display multiple panes and allow the user to adjust the size of the panes.

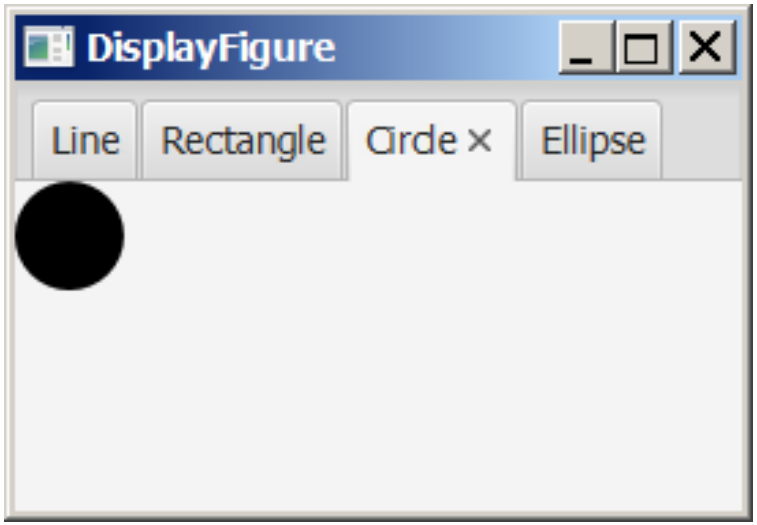




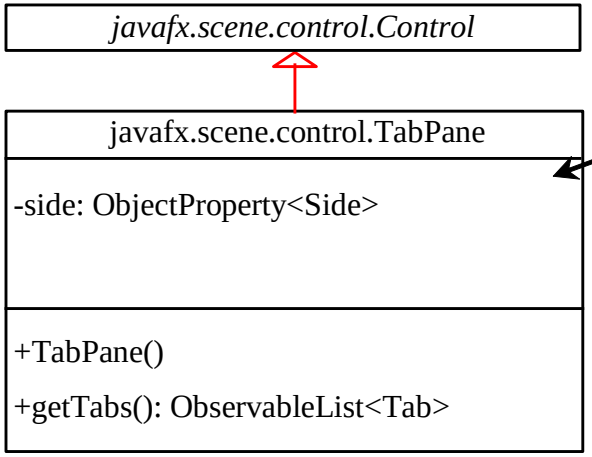
TabPane

- The TabPane class can be used to display multiple panes with tabs.





TabPane

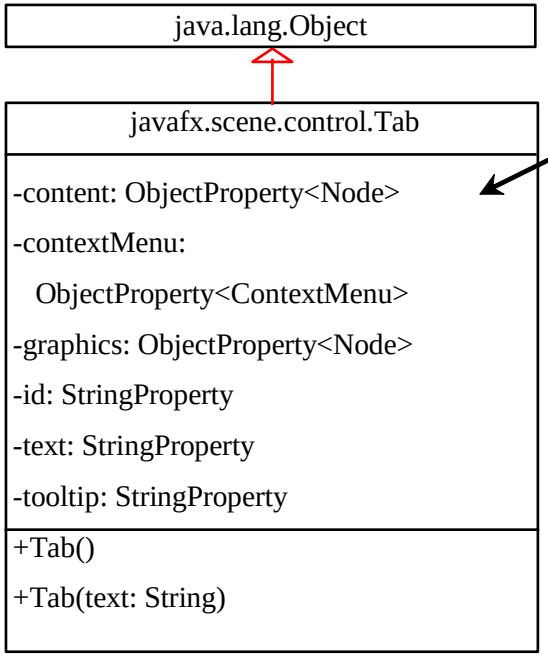


The `getter` and `setter` methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The position of the tab in the tab pane. Possible values are: Side.TOP, Side.BOTTOM, Side.LEFT, and Side.RIGHT (default: Side.TOP).

Creates a default tab pane.

Returns a list of tabs in this tab pane.



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 content associated with the tab.

The context menu associated with the tab.

The graphics in the tab.

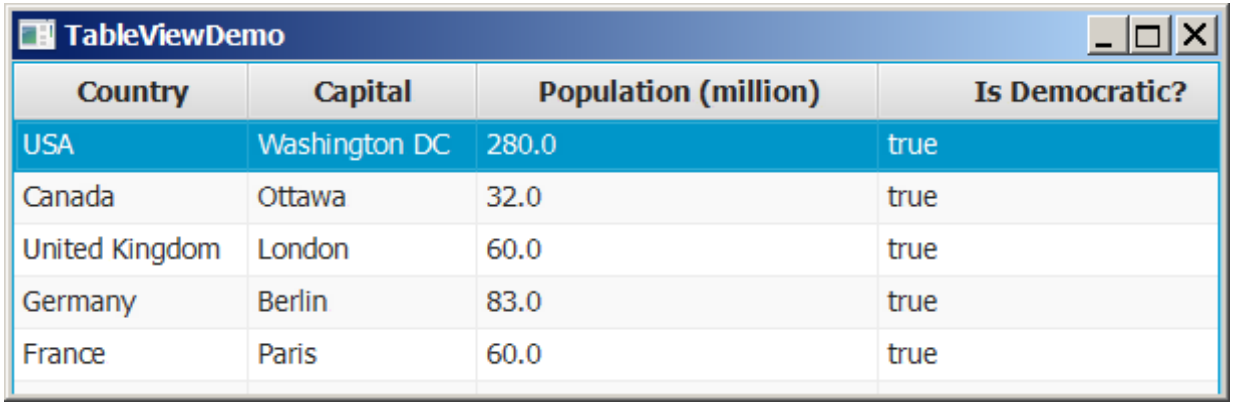
The id for the tab.

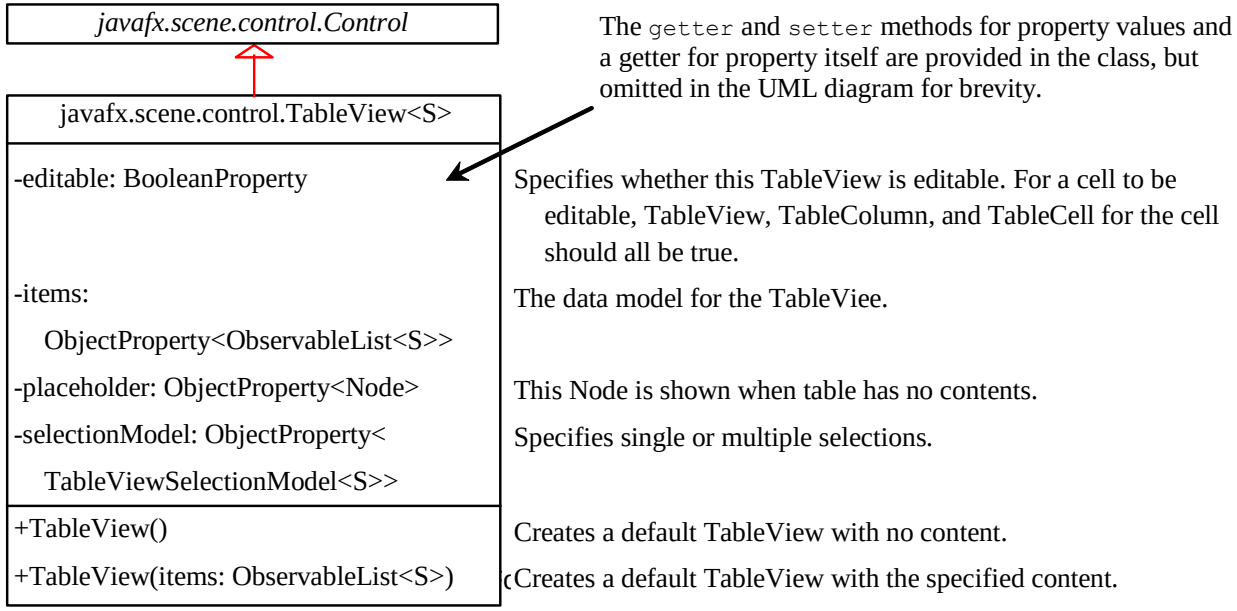
The text shown in the tab.

The tooltip associated with the tab.

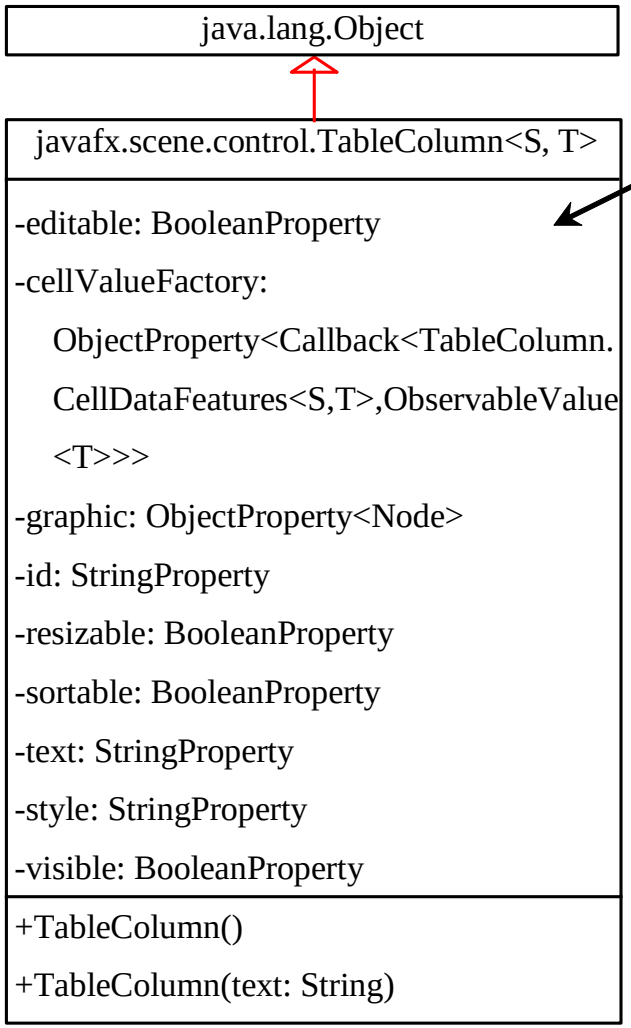
TableView

- You can display tables using the TableView class.





The TableColumn 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.

Specifies whether this TableColumn allows editing.

The cell value factory to specify how to populate all cells within a single column.

The graphic for this TableColumn.

The id for this TableColumn.

Indicates whether the column is resizable.

Indicates whether the column is sortable.

Text in the table column header.

Specify the CSS style for the column.

Specify whether the column is visible (default: true).

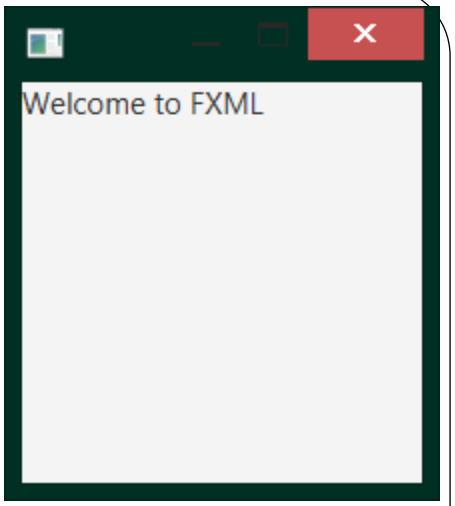
Creates a default TableColumn.

Creates a TableView with the specified header text.

FXML

- FXML is a declarative XML-based language created by Oracle Corporation for defining the user interface of a JavaFX 2.0 application.
  - It can be edited and created using the JavaFX Scene Builder 2 (downloaded separately from J2SE)
  - Create a new JavaFX project in Netbeans and you will get 3 files: an FXML file with the UI design, a main application .java file that loads the FXML and a controller for the event handlers for the UI Nodes.

FXML document:




```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="200"
xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8"
fx:controller="javafxapplication1.FXMLDocumentController">
    <children>
      <FlowPane prefHeight="200.0" prefWidth="200.0">
         <children>
              <Label fx:id="label" minHeight="16" minWidth="69"
                     text="Welcome to FXML" />
         </children>
      </FlowPane>
    </children>
</AnchorPane>
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
public class JavaFXApplication5 extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;
public class FXMLDocumentController implements Initializable {
    @FXML
    private Label label;
    @Override
    public void initialize(URL url, ResourceBundle rb) {
    }
}
```

HTML in JavaFX

- HTML intro.: the Internet Web pages format
- Example: **html_sample_01.html**

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

<span style="background-color:red; color:blue" >NOW?</span>

</body>
</html>
```

← This is the HTML tag. Every HTML page has one

← This is a heading

← This is a paragraph

HTML

- HTML is a language for describing web pages.
- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is a **markup** language
- A markup language is a set of markup **tags**
- The tags **describe** document content
- HTML documents contain HTML **tags** and plain **text**
- HTML documents are also called **web pages**

HTML

- HTML markup tags are usually called HTML tags
- HTML tags are keywords (tag names) surrounded by **angle brackets** like <html>
- HTML tags normally **come in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, with a **forward slash** before the tag name
- Start and end tags are also called **opening tags** and **closing tags**

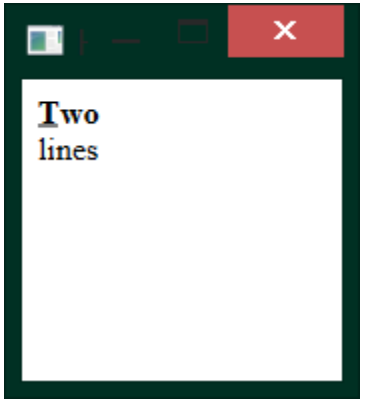
<tagname>content</tagname>

This is a paragraph.

HTML by Examples

- http://www.w3schools.com/html/html_examples.asp
- HTML links:
  - This is a link
  - It appears as: This is a link
- HTML images:
  - <img src="w3schools.jpg" width="104" height="142">
  - It appears as:

JavaFX with HTML



- **You can put HTML code in JavaFX:**

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.control.Button;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
public class HTMLDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    WebView browser = new WebView();
    WebEngine webEngine = browser.getEngine();
    webEngine.loadContent("<html><b><u>T</u>wo</b><br>lines</html>");
    StackPane root = new StackPane();
    root.getChildren().add(browser);
    Scene scene = new Scene(root, 100, 150);
    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
```

```
import org.w3c.dom.Document;
// ... get the document of the engine
Document doc = webEngine.getDocument();
// and the elements
import org.w3c.dom.Element;
... Element el = doc.getElementById("id1");
```

JavaFX with HTML

- **You can get the Document only when the asynchronized WebEngine had finished loading the page. That is,**
  `Document doc = webEngine.getDocument();`
  may be null if the page is not loaded yet.

  - **Solution: listen to the state of the WebEngine object to know when it is done loading:**

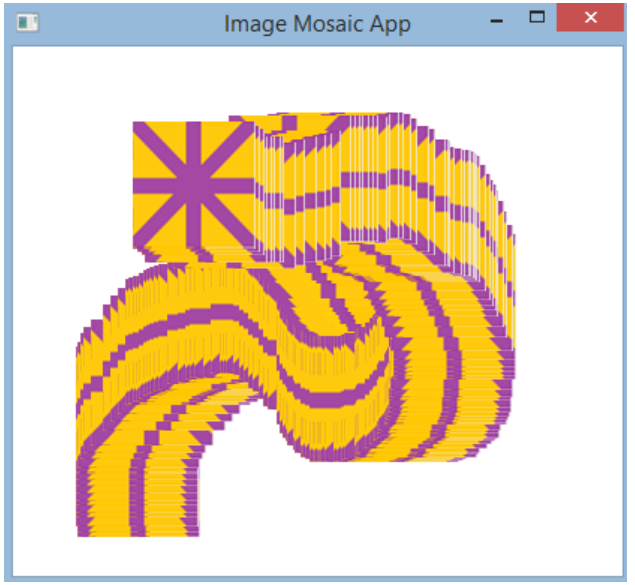
```
engine.getLoadWorker().stateProperty().addListener(
   (ObservableValue<? extends State> observable,
         State oldValue, State newValue)
         -> {
         if (newValue == State.SUCCEEDED)
         docManager.setStatsDoc(engine.getDocument());
});
```

javafx.scene.canvas.Canvas

- javafx.scene.canvas.Canvas is an image that can be drawn on using a set of graphics commands provided by a GraphicsContext.
- javafx.scene.canvas.GraphicsContext issues draw calls to a Canvas using a buffer:
  - each call pushes the necessary parameters onto the buffer where they will be later rendered onto the image of the Canvas node by the rendering thread at the end of a pulse.

```
Canvas canvas = new Canvas(250,250);
GraphicsContext gc =
    canvas.getGraphicsContext2D();
gc.fillRect(75,75,100,100);
```

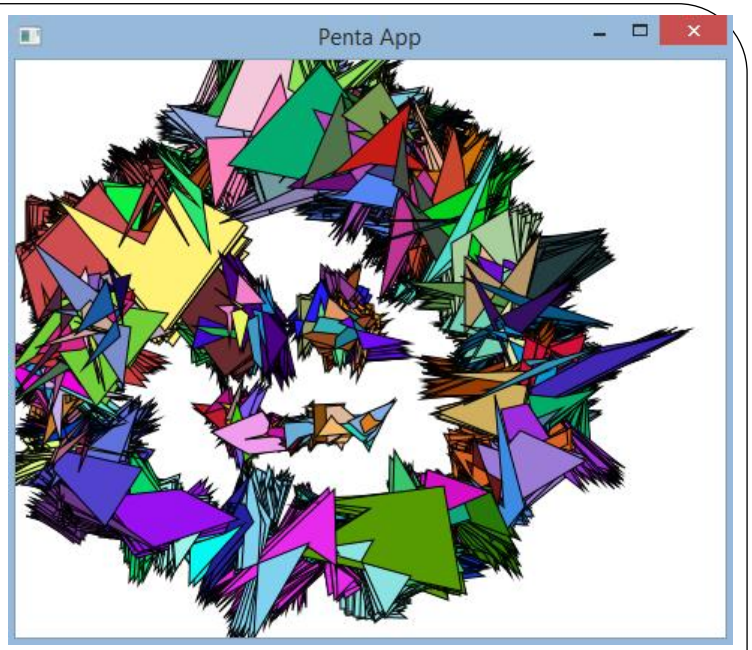
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.geometry.Point2D;
import javafx.geometry.Rectangle2D;
import javafx.scene.Group;
import javafx.scene.image.Image;
import javafx.stage.Screen;
import java.util.ArrayList;
import java.util.Iterator;
public class CanvasDemo extends Application {
    Stage primaryStage;
    Scene scene;
    Canvas canvas;
    GraphicsContext gc;
    Image logo1Image, logo2Image;
    ArrayList<Point2D> logo1Locations, logo2Locations;
    @Override
    public void start(Stage initPrimaryStage) {
        primaryStage = initPrimaryStage;
        initStage();
        initData();
        initGUI();
        initHandlers();
    }
```



```
public void initStage() {
    Screen screen = Screen.getPrimary();
    Rectangle2D bounds = screen.getVisualBounds();
    primaryStage.setX(bounds.getMinX());
    primaryStage.setY(bounds.getMinY());
    primaryStage.setWidth(bounds.getWidth());
    primaryStage.setHeight(bounds.getHeight());
}
public void initData() {
    logo1Locations = new ArrayList();
    logo2Locations = new ArrayList();
    logo1Image = new Image("file:images/logo1.png");
    logo2Image = new Image("file:images/logo2.png");
}
public void initGUI() {
    canvas = new Canvas();
    gc = canvas.getGraphicsContext2D(); // is graphics destination: monitor
    Group root = new Group();
    root.getChildren().add(canvas);
    scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
    canvas.setWidth(scene.getWidth());
    canvas.setHeight(scene.getHeight());
}
```

```
public void initHandlers() {
        canvas.setOnMouseClicked(mouseEvent -> {
            Point2D point = new Point2D(mouseEvent.getX(), mouseEvent.getY());
            if (!logo1Locations.contains(point)) {
                logo1Locations.add(point);
            }
            draw();
        });
        canvas.setOnMouseDragged(mouseEvent -> {
            Point2D point = new Point2D(mouseEvent.getX(), mouseEvent.getY());
            if (!logo2Locations.contains(point)) {
                logo2Locations.add(point);
            }
            draw();
        });
    }
    public void draw() {
        Iterator<Point2D> it = logo1Locations.iterator();
        while (it.hasNext()) {
            Point2D p = it.next();
            gc.drawImage(logo1Image, p.getX(), p.getY());
        }
        it = logo2Locations.iterator();
        while (it.hasNext()) {
            Point2D p = it.next();
            gc.drawImage(logo2Image, p.getX(), p.getY());
        }
    }
    public static void main(String[] args) {
        launch();
    }}
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.geometry.Rectangle2D;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.stage.Screen;
import java.util.ArrayList;
public class PentaApp extends Application {
    private Stage primaryStage;
    private Scene scene;
    private Canvas canvas;
    private GraphicsContext gc;
    private ArrayList<double[]> xPoints;
    private ArrayList<double[]> yPoints;
    private ArrayList<Color> colors;

    @Override
    public void start(Stage initPrimaryStage) {
        primaryStage = initPrimaryStage;
        initStage();
        initData();
        initGUI();
        initHandlers();
    }
```



```
public void initStage() {
    primaryStage.setTitle("Penta App");
    Screen screen = Screen.getPrimary(); // is graphics destination: monitor
    Rectangle2D bounds = screen.getVisualBounds();
    primaryStage.setX(bounds.getMinX());
    primaryStage.setY(bounds.getMinY());
    primaryStage.setWidth(bounds.getWidth());
    primaryStage.setHeight(bounds.getHeight());
}
public void initData() {
    xPoints = new ArrayList();
    yPoints = new ArrayList();
    colors = new ArrayList();
}
public void initGUI() {
    canvas = new Canvas();
    gc = canvas.getGraphicsContext2D();
    Group root = new Group();
    root.getChildren().add(canvas);
    scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
    canvas.setWidth(scene.getWidth());
    canvas.setHeight(scene.getHeight());
}
 public void initHandlers() {
    canvas.setOnMouseClicked(mouseEvent -> {
        if (mouseEvent.getClickCount() == 2) {
            xPoints.clear();
            yPoints.clear();
```

```
            colors.clear();
            gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
        }
    });
    canvas.setOnMouseDragged(mouseEvent -> {
        double x = mouseEvent.getX();
        double y = mouseEvent.getY();
        double[] xs = new double[5];
        double[] ys = new double[5];
        // CENTER
        xs[0] = x;
        ys[0] = y - (int) (Math.random() * 20) - 1;
        // TOP-RIGHT POINT
        xs[1] = x + (int) (Math.random() * 15) + 1;
        ys[1] = y - (int) (Math.random() * 10) - 1;
        // BOTTOM-RIGHT POINT
        xs[2] = x + (int) (Math.random() * 10) + 1;
        ys[2] = y + (int) (Math.random() * 15) + 1;
        // BOTTOM-LEFT POINT
        xs[3] = x - (int) (Math.random() * 10) - 1;
        ys[3] = y + (int) (Math.random() * 15) + 1;
        // TOP-LEFT POINT
        xs[4] = x - (int) (Math.random() * 15) - 1;
        ys[4] = y - (int) (Math.random() * 10) - 1;
        xPoints.add(xs);
        yPoints.add(ys);
        int r = (int) (Math.random() * 256);
        int g = (int) (Math.random() * 256);
        int b = (int) (Math.random() * 256);
        colors.add(Color.rgb(r, g, b));
        PentaApp.this.draw();
    });
}
```

```
    public void draw() {
        for (int i = 0; i < xPoints.size(); i++) {
            double[] xVertices = xPoints.get(i);
            double[] yVertices = yPoints.get(i);
            for (int j = 0; j < 5; j++) {
                xVertices[j] += (int) (Math.random() * 9) - 4;
                yVertices[j] += (int) (Math.random() * 9) - 4;
            }
            Color color = colors.get(i);
            gc.setFill(color);
            gc.fillPolygon(xVertices, yVertices, 5);
            gc.setStroke(Color.BLACK);
            gc.strokePolygon(xVertices, yVertices, 5);
        }
    }

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

}
```