

Graphical User Interfaces

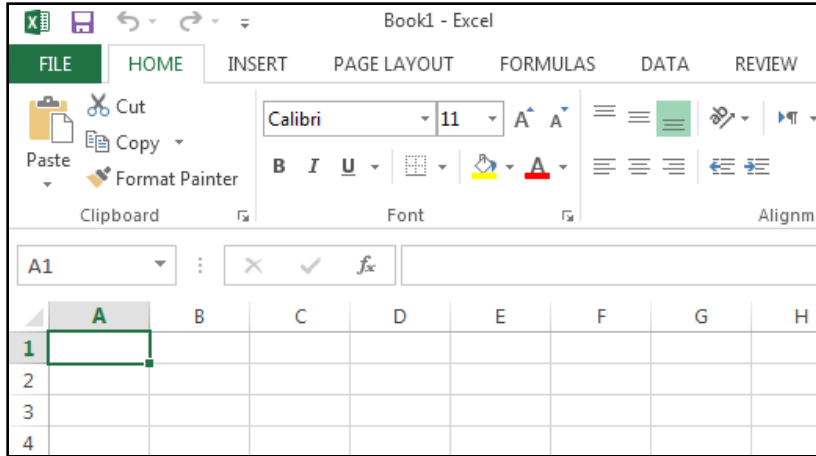
JavaFX GUI Basics

CSE114, Computer Science I

Stony Brook University

<http://www.cs.stonybrook.edu/~cse114>

GUI Examples

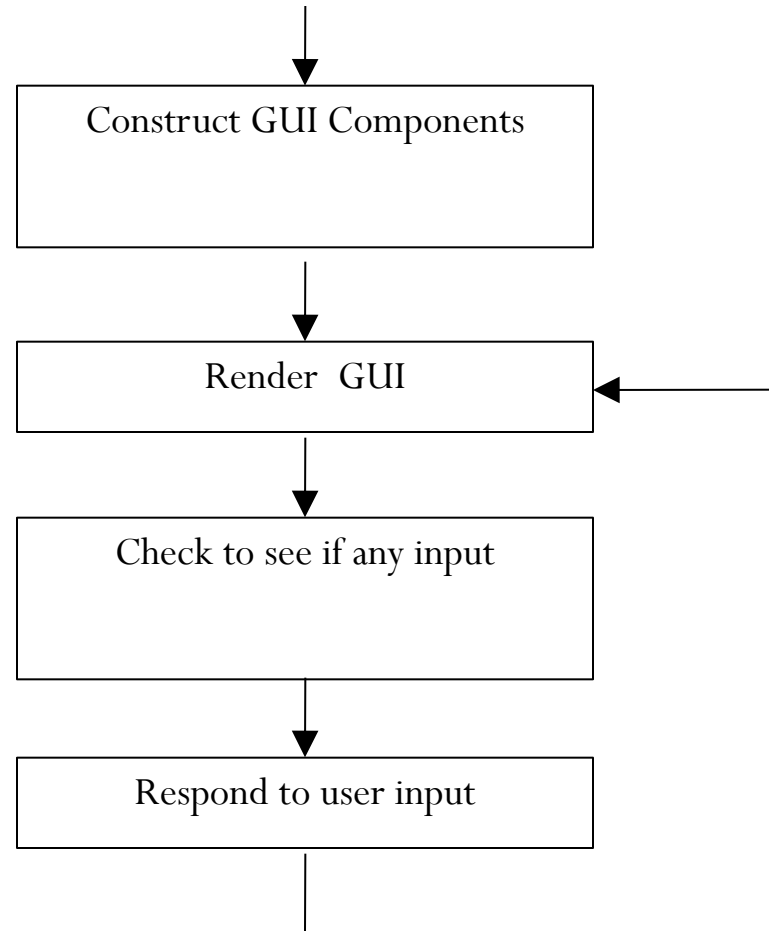


GUI

- Graphical User Interface (GUI)
 - provides user-friendly human interaction
- Building Java GUIs require use of multiple frameworks:
 - **JavaFX (part of JSE 8, 2014)**
 - An old framework would use:
 - Java's GUI component Libraries
 - **javax.swing.***
 - Java's Event Programming Libraries
 - **java.awt.event.***
 - **javax.swing.event.***
 - Java's Graphics Programming Libraries
 - **java.awt.***
 - **java.awt.geom.***

How do GUIs work?

- They loop and respond to events



Example: a mouse click on a button

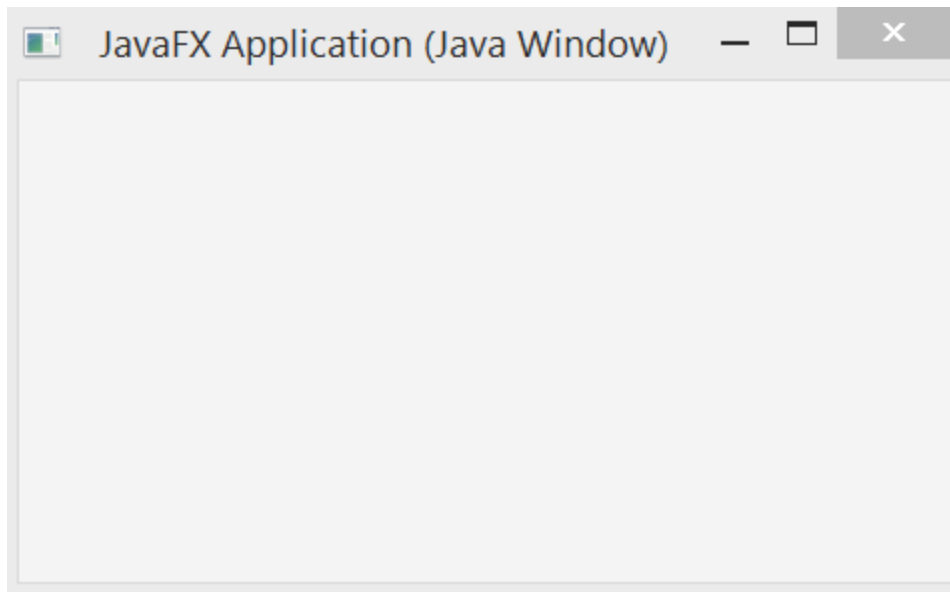
- Operating System recognizes mouse click
 - determines which window it was inside
 - notifies that program
- Program runs in loop
 - checks input buffer filled by OS
 - if it finds a mouse click:
 - determines which component in the program
 - if the click was on a relevant component
 - respond appropriately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

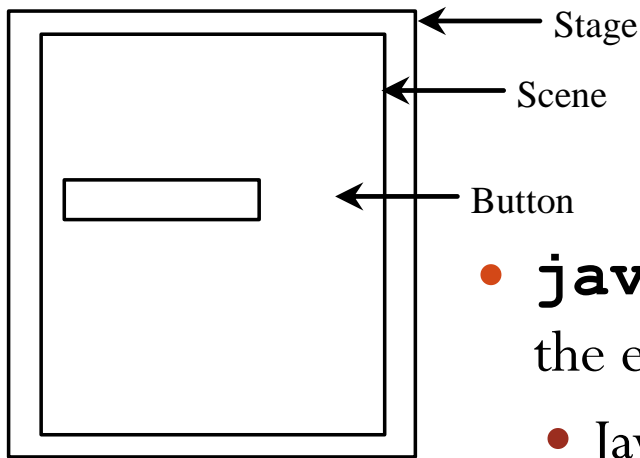


```
public class JavaFXApplication1
    extends Application {
    @Override
    public void start(Stage
        primaryStage) {
        Scene scene =
            new Scene(root, 300, 250);
        primaryStage.setTitle("JavaFX
            Application (Java Window)");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(
        String[] args) {
        launch(args);
    }
}
```

JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications in JDK8.
- When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT).
 - AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
 - In addition, AWT is prone to platform-specific bugs.
- The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as Swing components.
 - **Swing components are painted directly on canvases using Java code.**
 - Swing components depend less on the target platform and use less of the native GUI resource.
- **With the release of Java 8, Swing is replaced by a completely new GUI platform: 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
    }

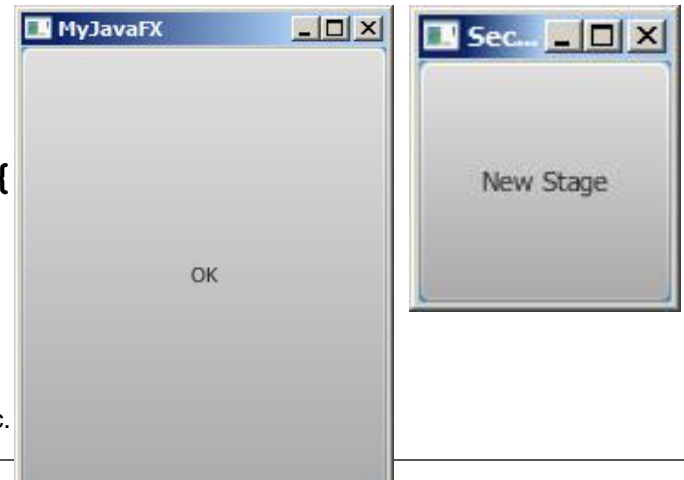
    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```



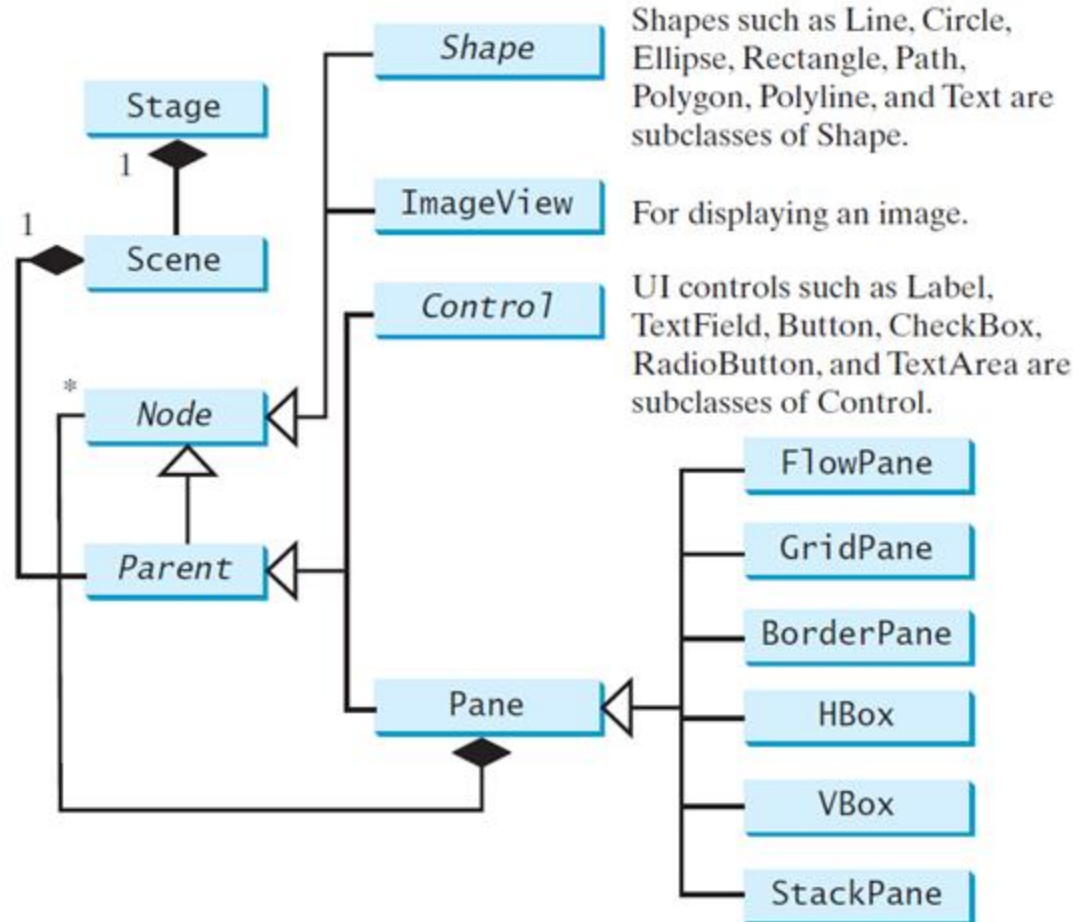
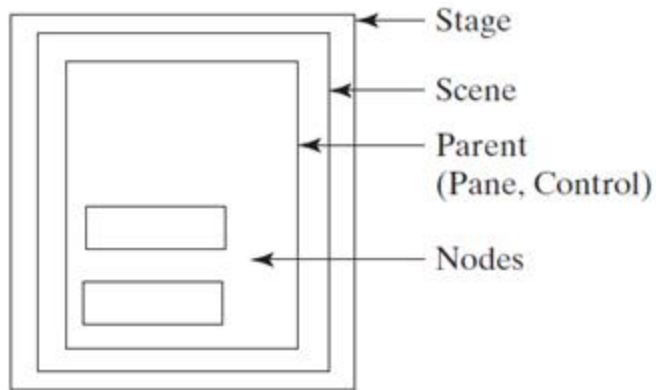
```
// Multiple stages can be added beside the primaryStage
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

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
    }

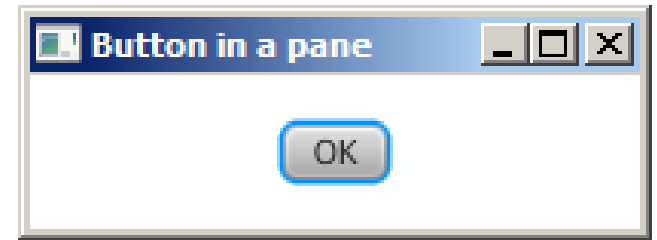
    public static void main(String[] args) {
        launch(args);
    }
}
```



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;
```

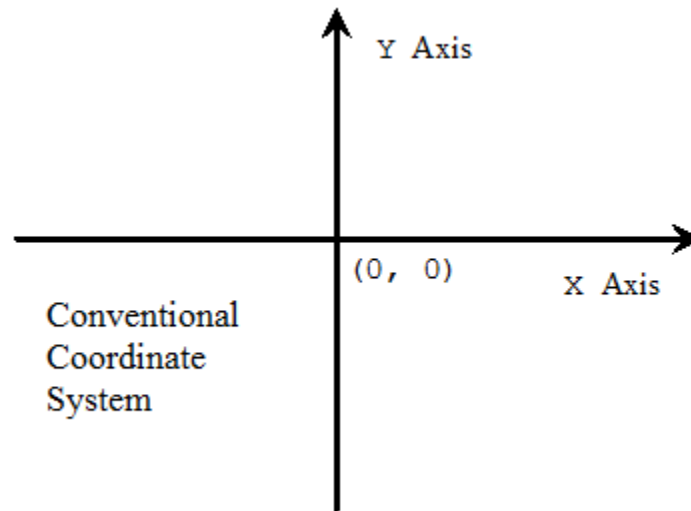


```
public class ButtonInPane extends Application {

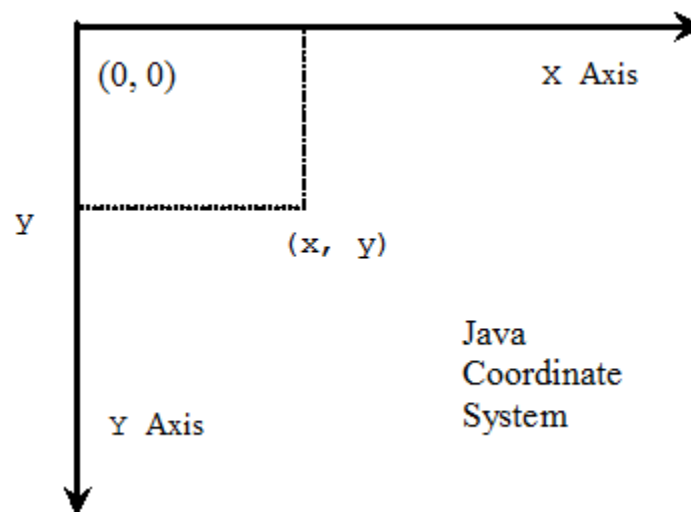
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        StackPane pane = new StackPane();
        pane.getChildren().add(new Button("OK"));
        Scene scene = new Scene(pane, 200, 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

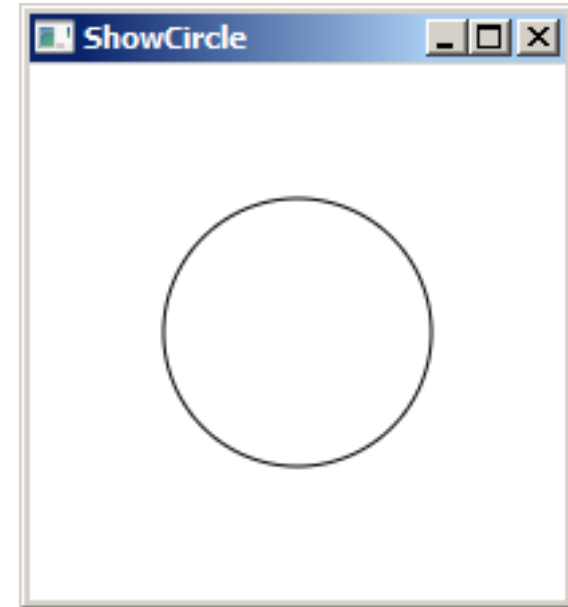


Circle in a Pane

```
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;

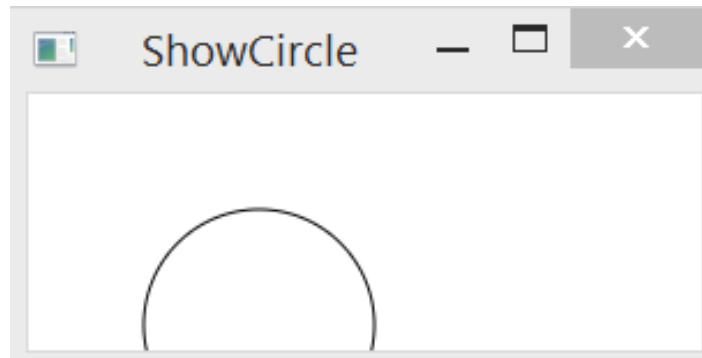
public class ShowCircle extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a circle and set its properties
        Circle circle = new Circle();
        circle.setCenterX(100);
        circle.setCenterY(100);
        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

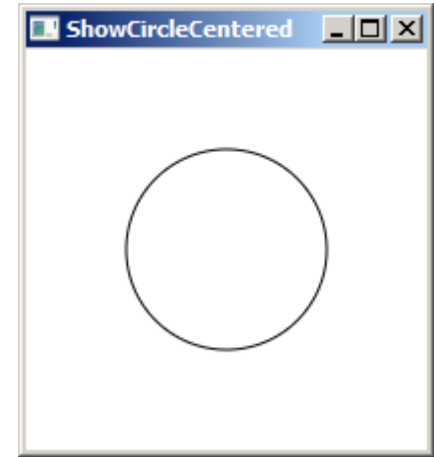
- JavaFX introduces a new concept called *binding property* that enables a target object to be bound to a source object.
 - If the value in the source object changes, the target property is also changed automatically.
 - The target object is simply called a binding object or a binding property.
- Resizing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

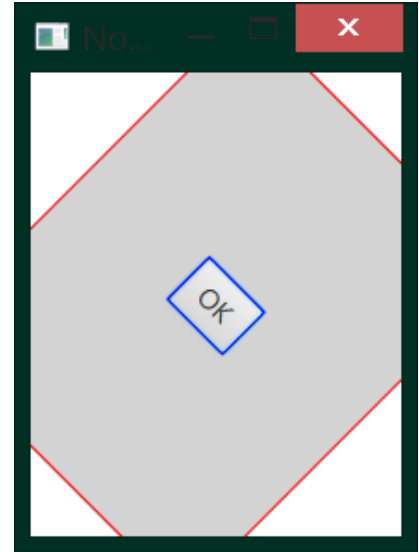
Output:
d1 is 2.0 and d2 is 2.0

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
    }

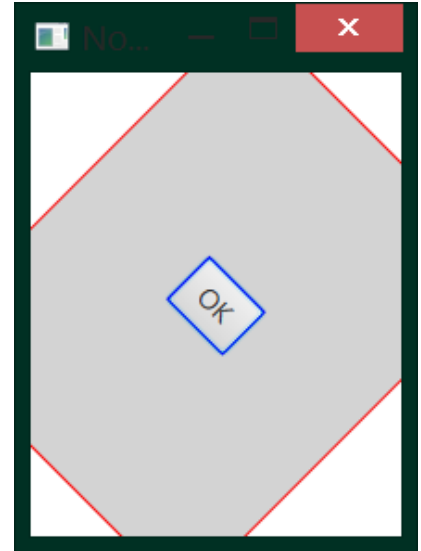
    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```



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

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

javafx.scene.paint.Color

```
-red: double
-green: double
-blue: double
-opacity: double

+Color(r: double, g: double, b: double, opacity: double)
+brighter(): Color
+darker(): Color
+color(r: double, g: double, b: double): Color
+color(r: double, g: double, b: double, opacity: double): Color
+rgb(r: int, g: int, b: int): Color
+rgb(r: int, g: int, b: int, opacity: double): Color
```

The red value of this Color (between 0.0 and 1.0).

The green value of this Color (between 0.0 and 1.0).

The blue value of this Color (between 0.0 and 1.0).

The opacity of this Color (between 0.0 and 1.0).

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

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

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

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

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

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

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

Helper classes: The Font Class

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

javafx.scene.text.Font

-size: double
-name: String
-family: String

+Font(size: double)
+Font(name: String, size: double)
+font(name: String, size: double)
+font(name: String, w: FontWeight, size: double)
+font(name: String, w: FontWeight, p: FontPosture, size: double)
+getFamilies(): List<String>
+getFontNames(): List<String>

The size of this font.

The name of this font.

The family of this font.

Creates a Font with the specified size.

Creates a Font with the specified full font name and size.

Creates a Font with the specified name and size.

Creates a Font with the specified name, weight, and size.

Creates a Font with the specified name, weight, posture, and size.

Returns a list of font family names.

Returns a list of full font names including family and weight.

The Image and ImageView Classes

`javafx.scene.image.Image`

`-error: ReadOnlyBooleanProperty`
`-height: ReadOnlyBooleanProperty`
`-width: ReadOnlyBooleanProperty`
`-progress: ReadOnlyBooleanProperty`

`+Image(filenameOrURL: String)`

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

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.

`javafx.scene.image.ImageView`

`-fitHeight: DoubleProperty`
`-fitWidth: DoubleProperty`
`-x: DoubleProperty`
`-y: DoubleProperty`
`-image: ObjectProperty<Image>`

`+ImageView()`
`+ImageView(image: Image)`
`+ImageView(filenameOrURL: String)`

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 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.**

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

FlowPane

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

`javafx.scene.layout.FlowPane`

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

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

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

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

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

Creates a default `FlowPane`.

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

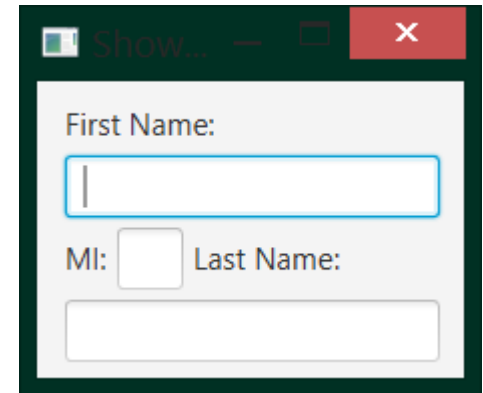
Creates a `FlowPane` with a specified orientation.

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

```

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
+setHorizontalAlignment(child: Node, value: HPos): void
+setVerticalAlignment(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.

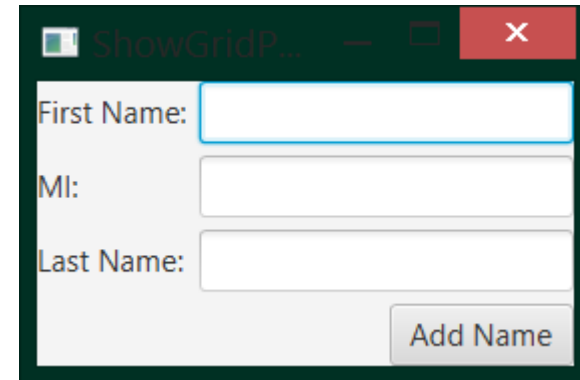
Sets the vertical alignment for the child in the cell.

```

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

`javafx.scene.layout.BorderPane`

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

`+BorderPane()`

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

The 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 node placed in the top region (default: `null`).

The node placed in the right region (default: `null`).

The node placed in the bottom region (default: `null`).

The node placed in the left region (default: `null`).

The node placed in the center region (default: `null`).

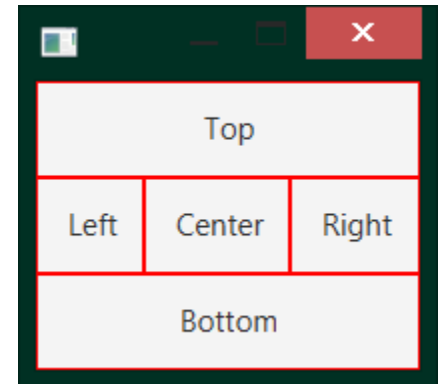
Creates a `BorderPane`.

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

```

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.

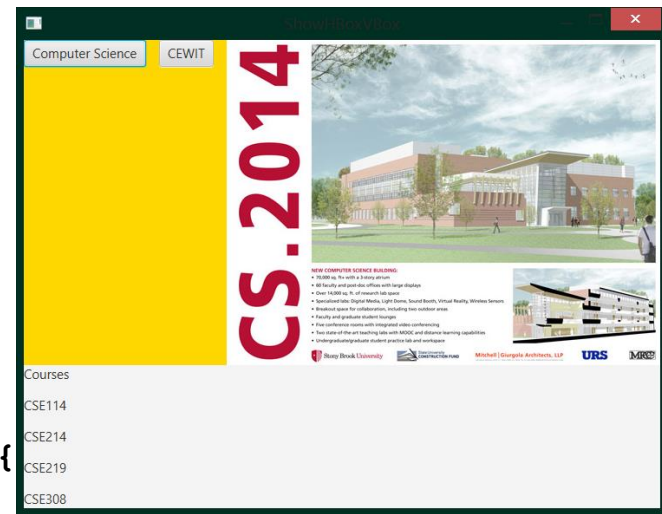
Sets the margin for the node in the pane.

```

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();
    }
}

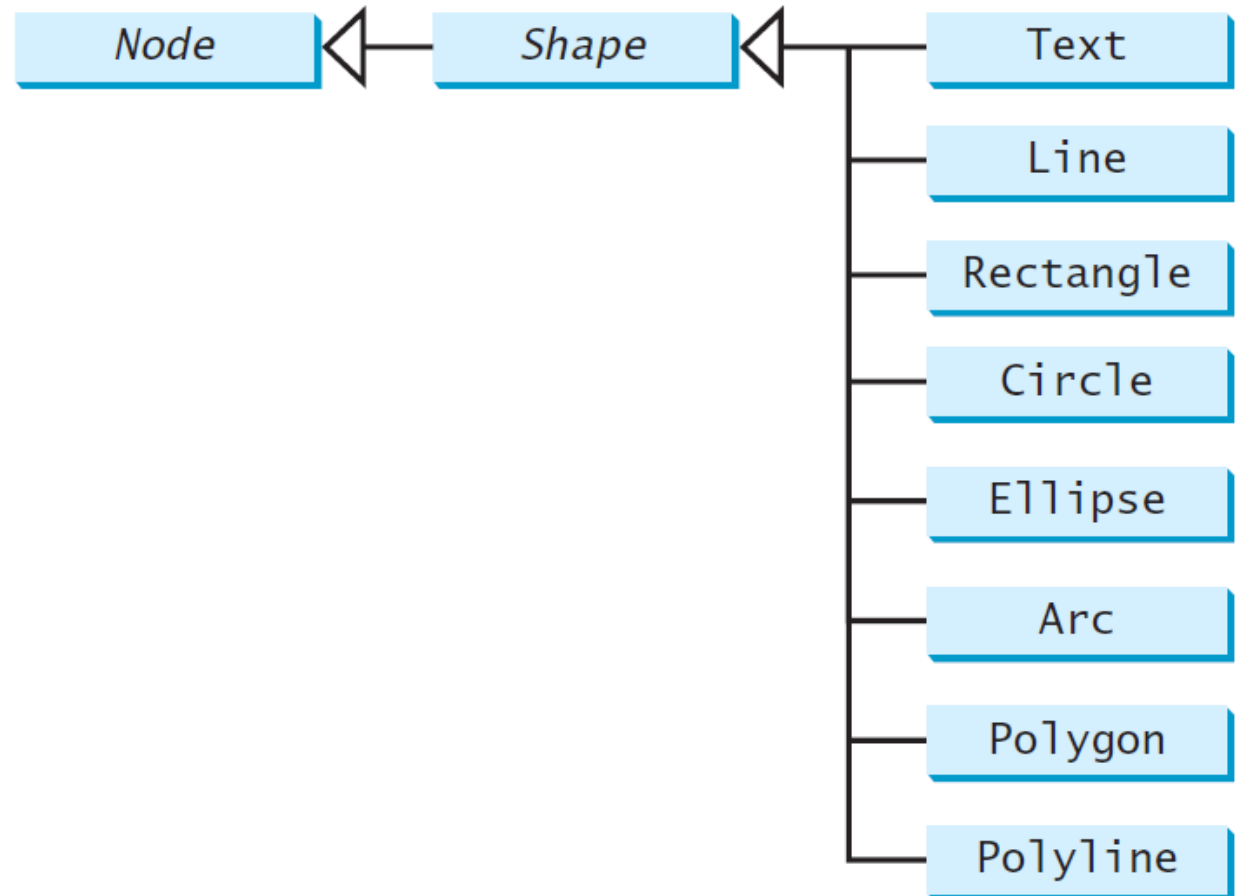
```

(c) Paul Fodor and Pearson Inc.



Shapes

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



Text

`javafx.scene.text.Text`

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

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

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

Defines the text to be displayed.

Defines the x-coordinate of text (default 0).

Defines the y-coordinate of text (default 0).

Defines if each line has an underline below it (default `false`).

Defines if each line has a line through it (default `false`).

Defines the font for the text.

Creates an empty Text.

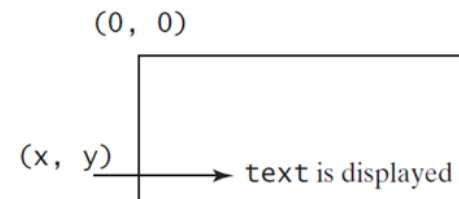
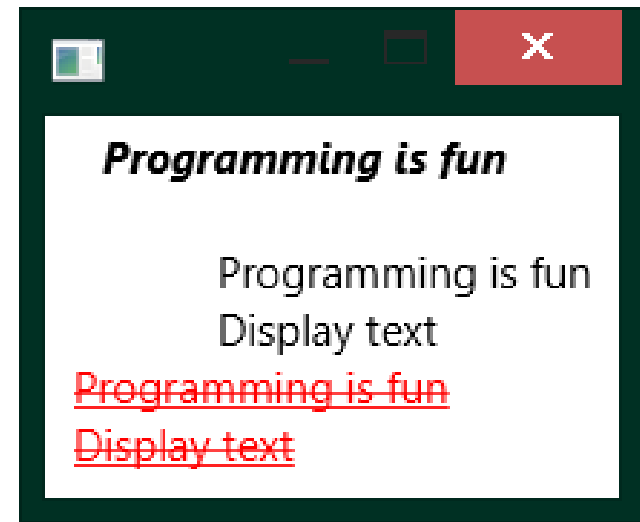
Creates a Text with the specified text.

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

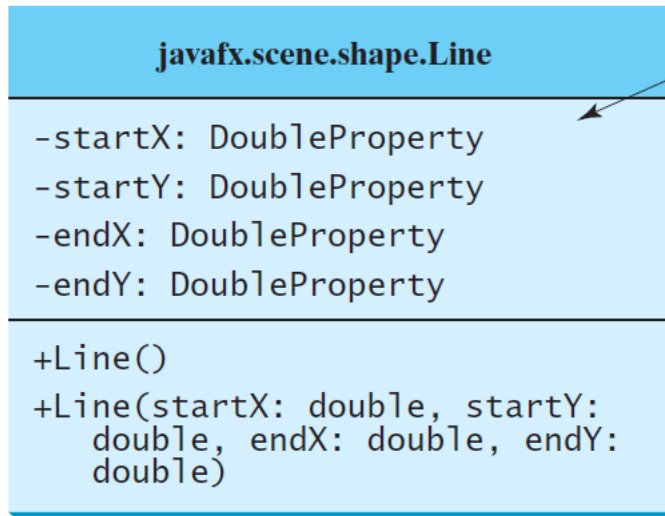
```

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);
        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.

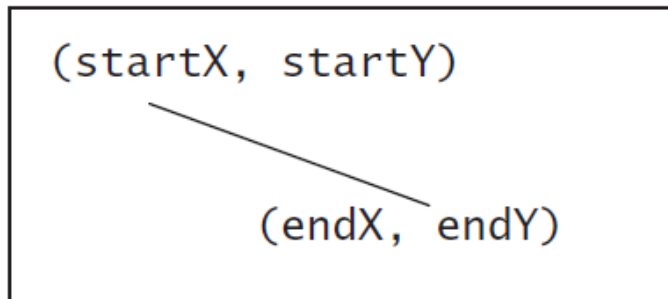
The x-coordinate of the start point.
The y-coordinate of the start point.
The x-coordinate of the end point.
The y-coordinate of the end point.

Creates an empty `Line`.

Creates a `Line` with the specified starting and ending points.

(0, 0)

(getWidth(), 0)



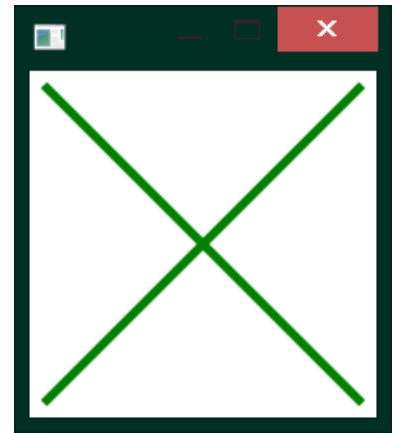
(0, getHeight())

(getWidth(), getHeight())

```

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

javafx.scene.shape.Rectangle

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

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

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

The 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).

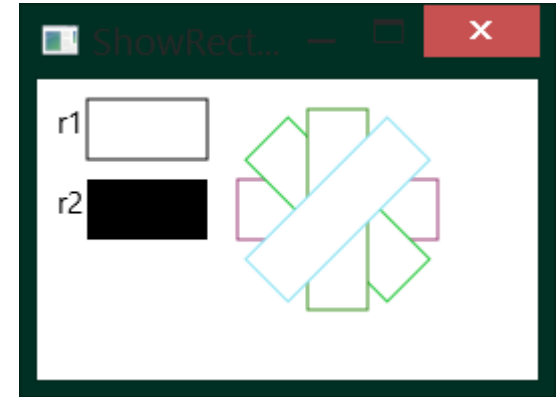
Creates an empty Rectangle.

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

`javafx.scene.shape.Circle`

`-centerX: DoubleProperty`
`-centerY: DoubleProperty`
`-radius: DoubleProperty`

`+Circle()`
`+Circle(x: double, y: double)`
`+Circle(x: double, y: double,
radius: double)`

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

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

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

Ellipse

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)

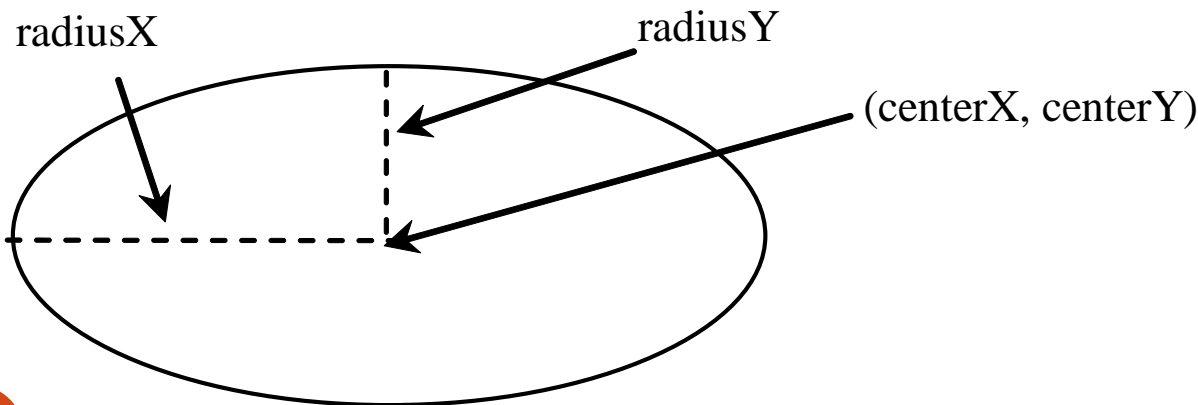
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The x-coordinate of the 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).

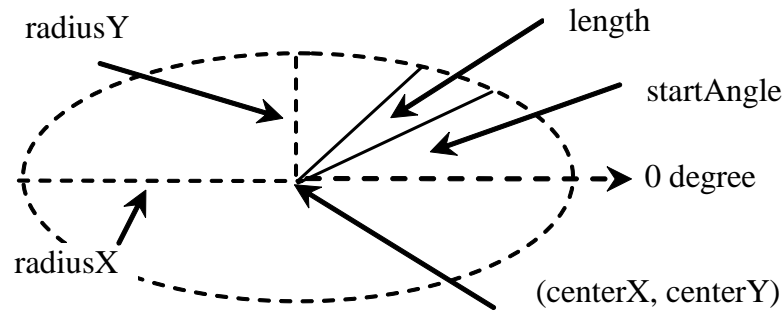
Creates an empty `Ellipse`.

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.

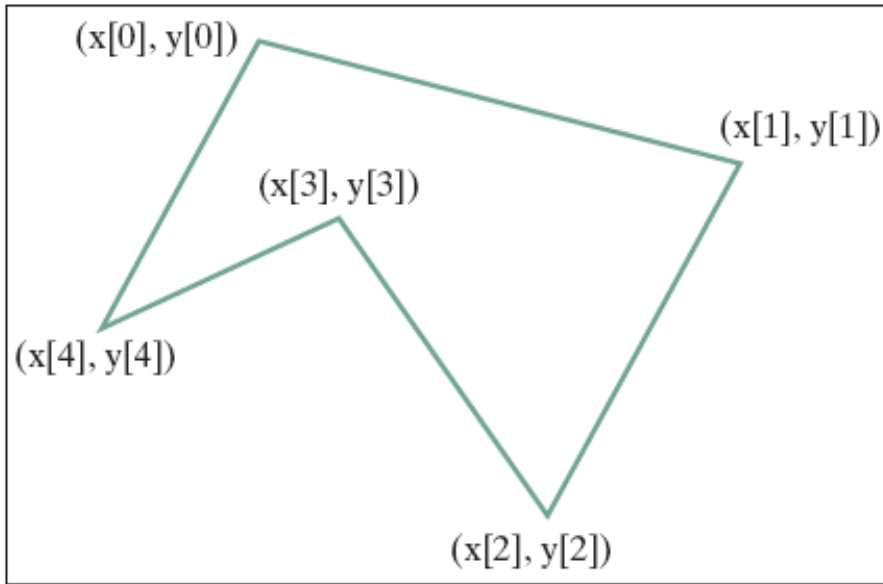
The angular extent of the arc in degrees.

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

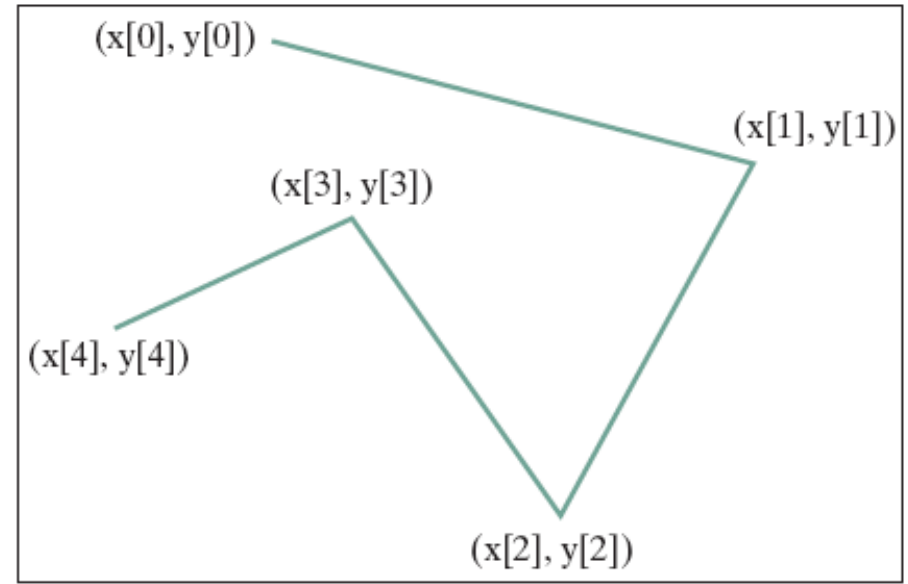
Creates an empty Arc.

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.