GUI development and JavaFX Basics

Dr. Abdallah Karakra |Comp 2311 | Masri504

12/06/2023

Index

Chapter 14

- All Sections (14.1 – 14.12)

CHAPTER 14

JavaFX Basics

JavaFX vs Swing vs AWT

- Java was first released with GUI support in something called the Abstract Windows Toolkit (AWT)
- AWT wasn't bad, but it had some limitations, and some particular problems with how it was implemented on some platforms (in other words, AWT is fine for developing simple GUIs, but not good for developing comprehensive GUI projects. In addition, AWT is prone to platform-specific bugs.)
- Ultimately, AWT (which still exists, but isn't used much anymore) was replaced by a new library called Swing, which was more versatile, more robust, and more flexible.

JavaFX vs Swing vs AWT

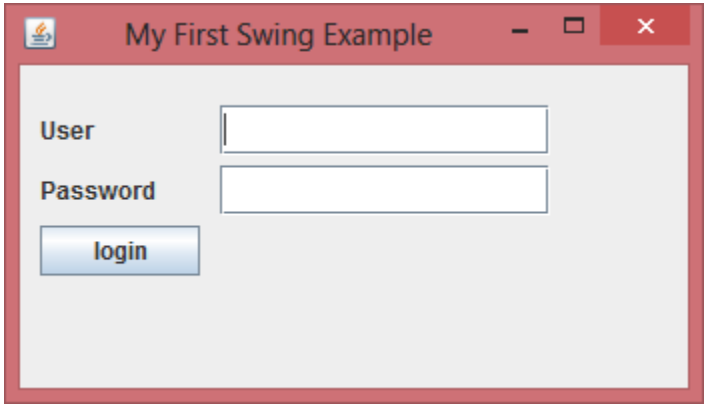
- Swing was designed primarily for use in desktop applications (although you could do some web-based things with it, too).
- Swing has now been replaced by a completely new GUI library called JavaFX
- You can still use Swing (for the foreseeable future), but Oracle isn't going to develop it any further – it's essentially a dead-end technology
- Java has replaced Swing with JavaFX
- How long until JavaFX is replaced by something else? Nobody knows; probably many years

JavaFX is a newer framework for developing Java GUI programs

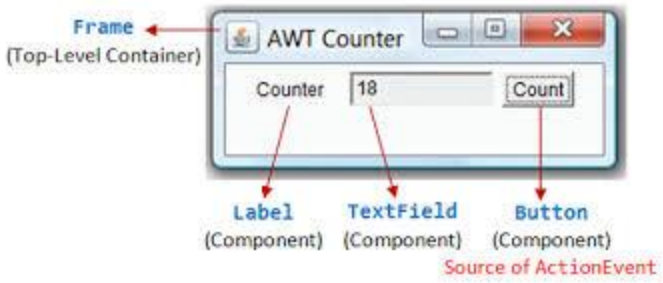
JavaFX vs. Swing & AWT

JavaFX example

Swing example



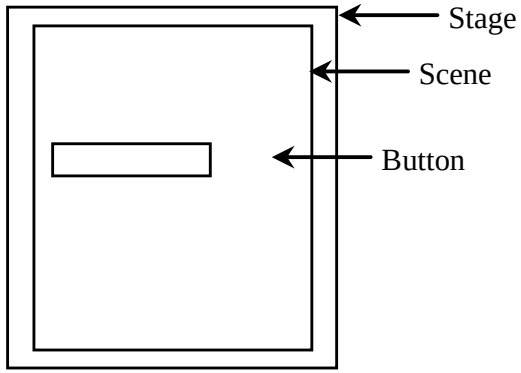
AWT example



The Basic Structure of a JavaFX Program

Basic Structure of JavaFX

- Every JavaFX program is defined in a class that extends abstract Application class in JavaFX, **javafx.application.Application**
- Override the start(Stage) method: The start method normally places UI controls in a scene and displays the scene in a stage



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

- Stage, Scene, and Nodes

Basic Structure of JavaFX

```
public class MyProgram
{
    // Body of class
}
```

Becomes:

```
import javafx.application.Application;
…
public class MyProgram extends Application
{
    // Body of class
}
```

The abstract javafx.application.Application class defines the essential framework for writing JavaFX programs.

Our First JavaFX Program

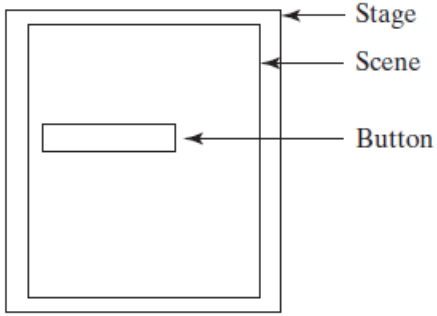
- Our first JavaFX program will open a window, whose title bar will display "MyJavaFX", and which will have a (huge) button in the middle labeled "OK"
- First, the result, and then the code:

Our First JavaFX Program


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

public class MyJavaFX extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a button and place it in the scene
    Button btOK = new Button("OK"); //Create a button
    Scene scene = new Scene(btOK, 200, 250); //Create a Scene
    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);
  }
}
```

Our First JavaFX Program

- In JavaFX, the stage is the window our code runs in
- Since every GUI application, by definition, involves a window with the UI, we get the primaryStage by default when the application launches.
- Our applications are not limited to a single stage
- The code to setup this two-stage UI is on the next slide



Our First JavaFX Program

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

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

    Stage stage = new Stage(); // Create a new stage
    stage.setTitle("Second Stage"); // Set the stage title
    // Set a scene with a button in the stage
    stage.setScene(new Scene(new Button("New Stage"), 200, 250));
    stage.show(); // Display the stage
  }

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

Our First JavaFX Program

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

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

    Stage stage = new Stage(); // Create a new stage
    stage.setTitle("Second Stage"); // Set the stage title
    // Set a scene with a button in the stage
    stage.setScene(new Scene(new Button("New Stage"), 200, 250));
    stage.show(); // Display the stage
  }

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

Every JavaFX program extends javafx.application.Application

The main class overrides the start method defined in javafx.application.Application and JVM constructs an instance of the class and invoke the start method.

The first Scene object is created using the constructor – specifies width and height and places button in scene

The first Stage object is automatically created by JVM when the app is launched

Name the first Stage, set the scene in the stage, and display the stage.

A new stage is created.

Name the second Stage, set the scene in the stage, places button in Scene, and display the stage.

The launches JavaFX app. Identical main method for all every JavaFX app.

Our First JavaFX Program

- By default, stages (windows) are resizeable.

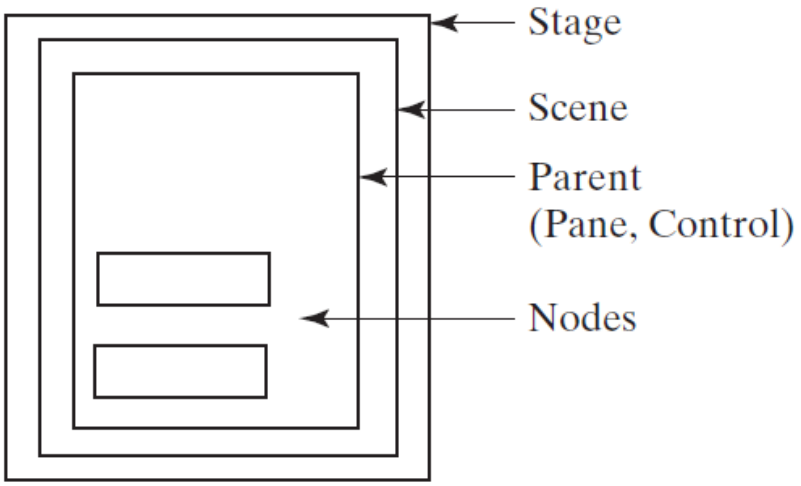
- Note that we have minimize and maximize buttons

-If we want our stage to be of fixed size (i.e., not resizeable), we can set that property with `stage.setResizeable(false)`

Panes, UI Controls, and Shapes

Panes, UI Controls, and Shapes

- In the previous examples, we put the button directly on the scene, which centered the button and made it occupy the entire window.
- Rarely is this what we really want to do
- One approach is to specify the size and location of each UI element (like the buttons)
- A better solution is to put the UI elements (known as nodes) into containers called panes, and then add the panes to the scene.



Panes, UI Controls, and Shapes

- The following slide shows the code to create this version of the same UI, with a single button inside a pane (so that the button doesn't occupy the whole stage).

- It uses a StackPane (which we'll discuss later).

- In order to add something to a pane, we need to access the list of things IN the pane, much like an ArrayList.

- The new item we'll add will be a new child of the pane, so we're adding it to the list of the pane's children

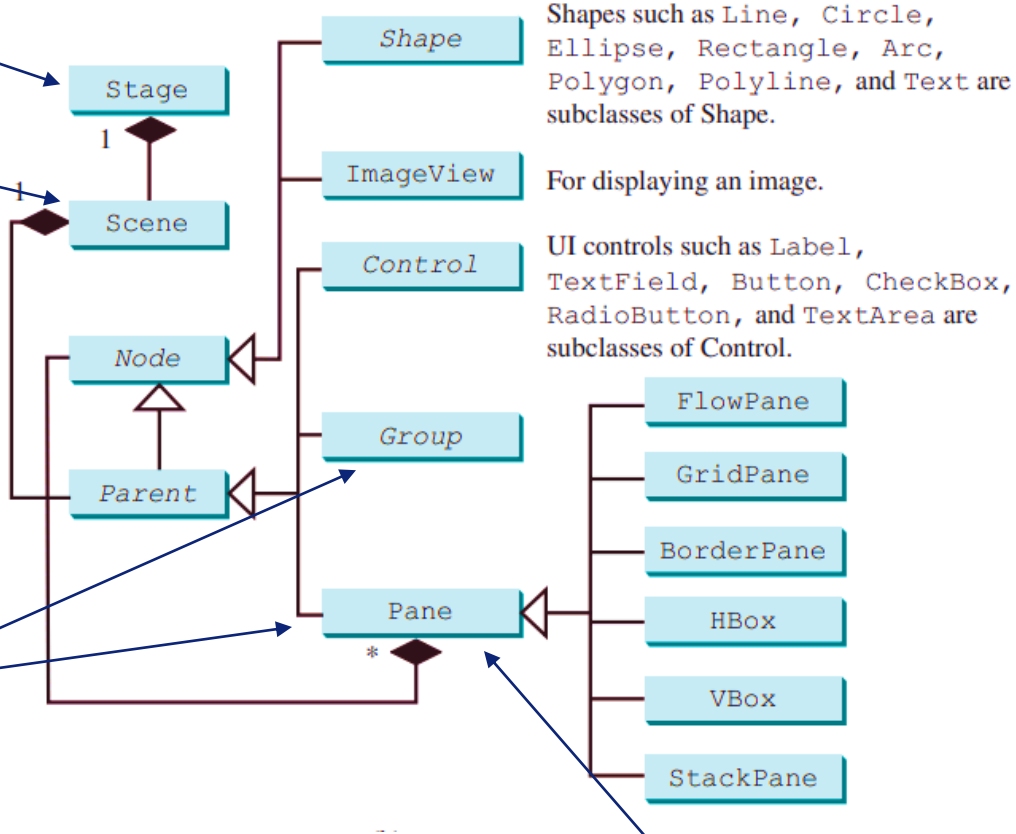
Button In Pane


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

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

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

Panes, UI Controls, and Shapes

(1) One Scene per Stage

(2) A Scene can contain a **Control, Group or Pane**, but not a **Shape or an ImageView**.



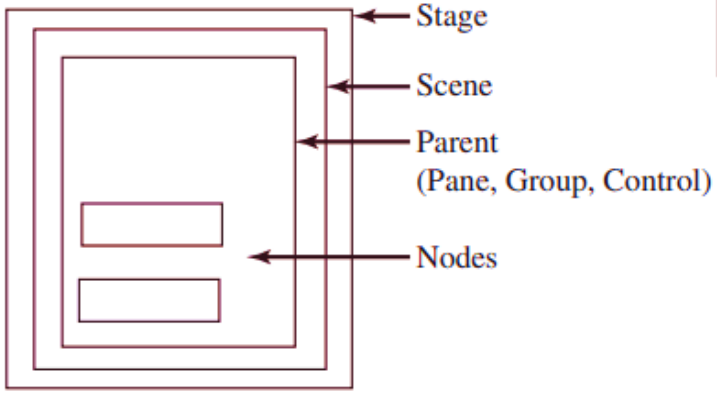
(3) **A Pane or Group** can contain any subtype of Node

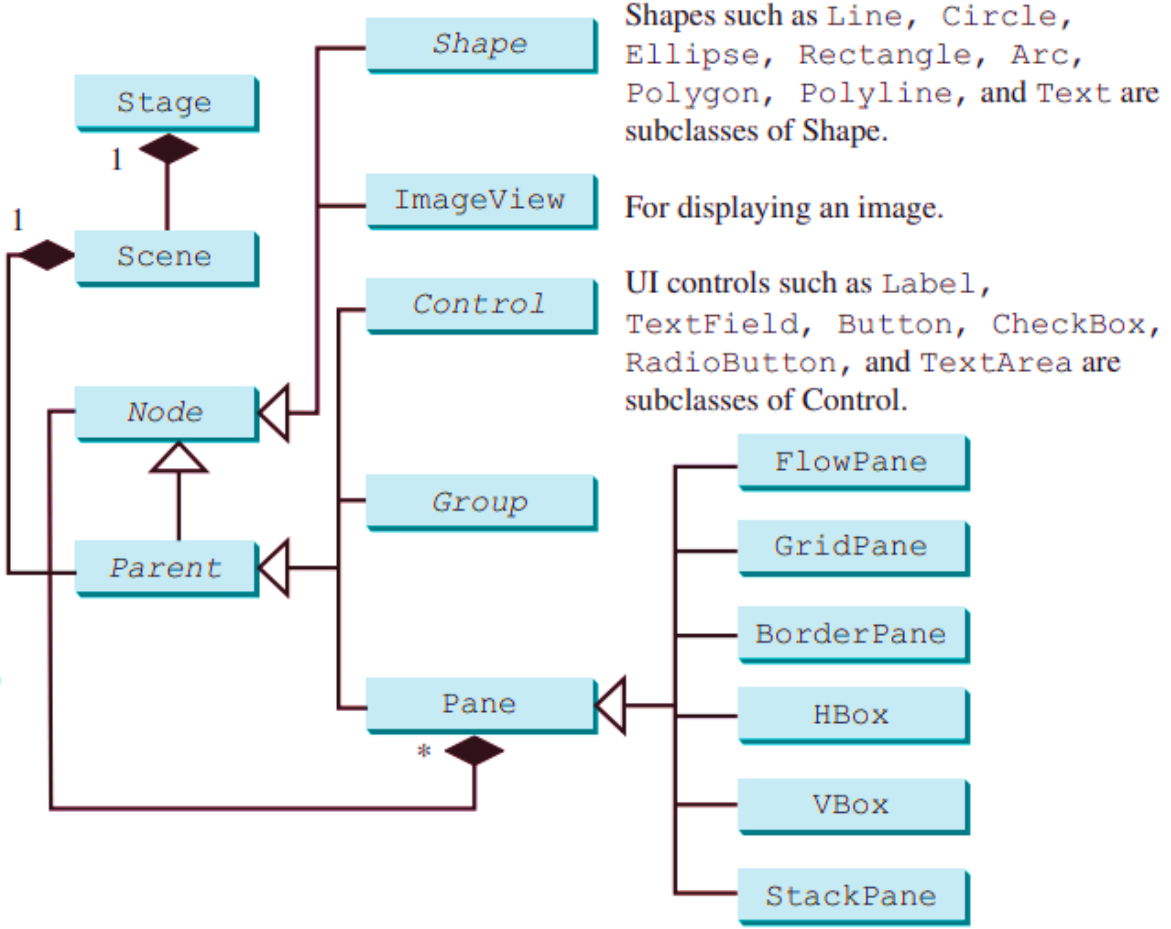
To create a Scene use one of the following:
Scene(Parent, width, height) or Scene(Parent).

(4) Will cover the various types of Panes later

Panes, UI Controls, and Shapes

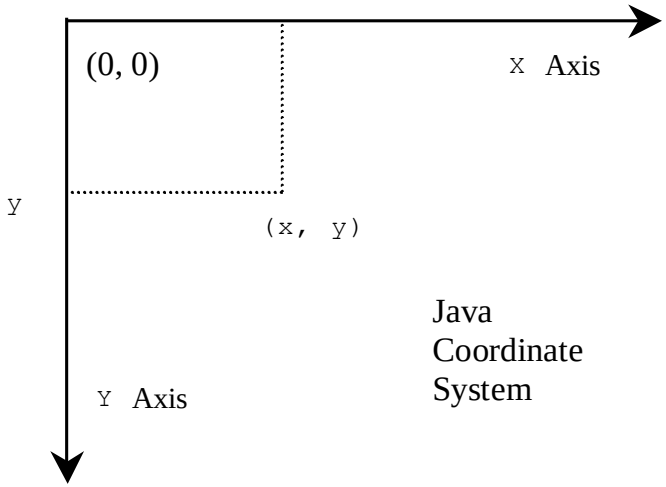
1. Extend `Application`
2. Override `start(Stage)`
3. Create Nodes
4. Place Nodes in a *`Parent`*
5. Place the *`Parent`* in the `Scene`
6. Place the `Scene` on `Stage`
7. Show `Stage`

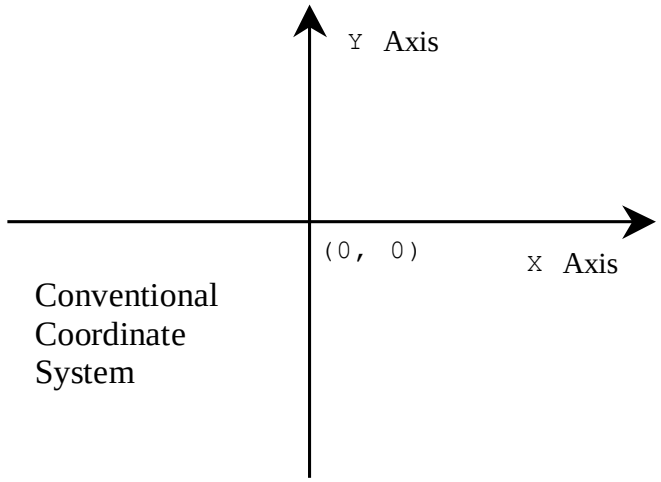




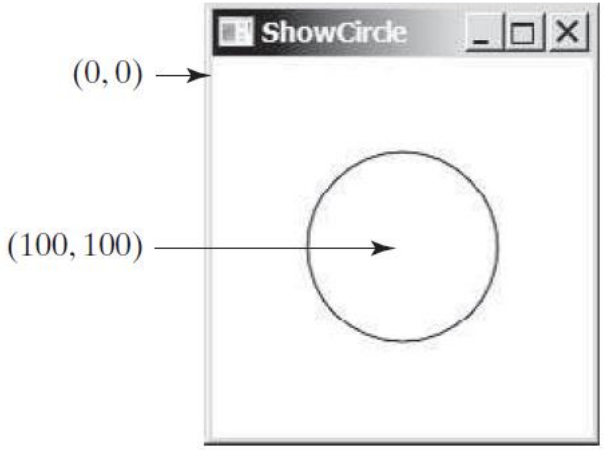
Display a Shape

This example displays a circle in the center of the pane.



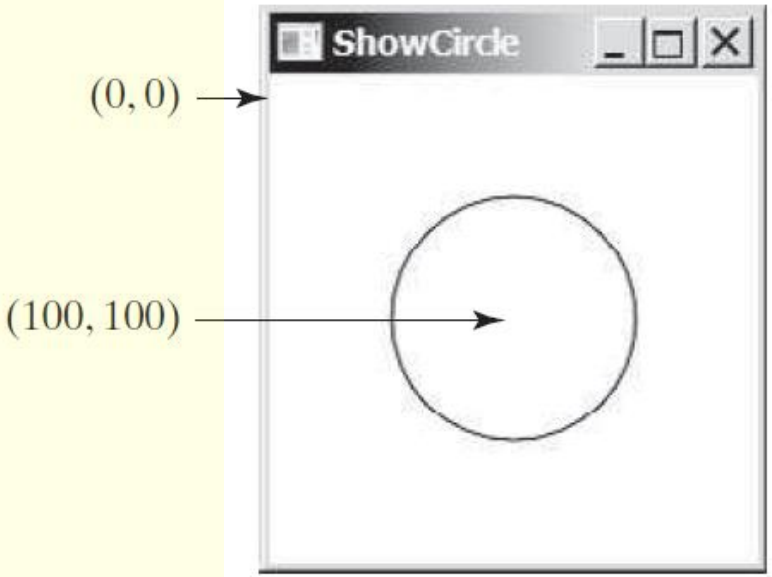


The top-left corner of a scene is *always* (0, 0), and the (positive) X-axis goes to the right, and the (positive) Y-axis goes down.



Display a Shape: `Circle` at the Center of a `Pane`

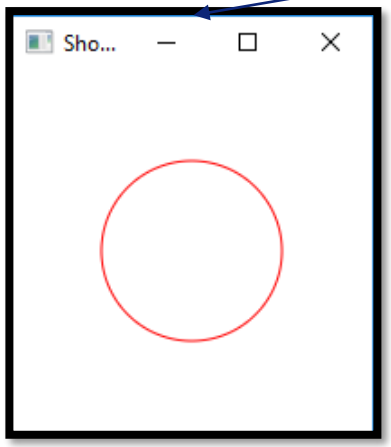
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

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(Color.WHITE);

    // 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
  }
}
```

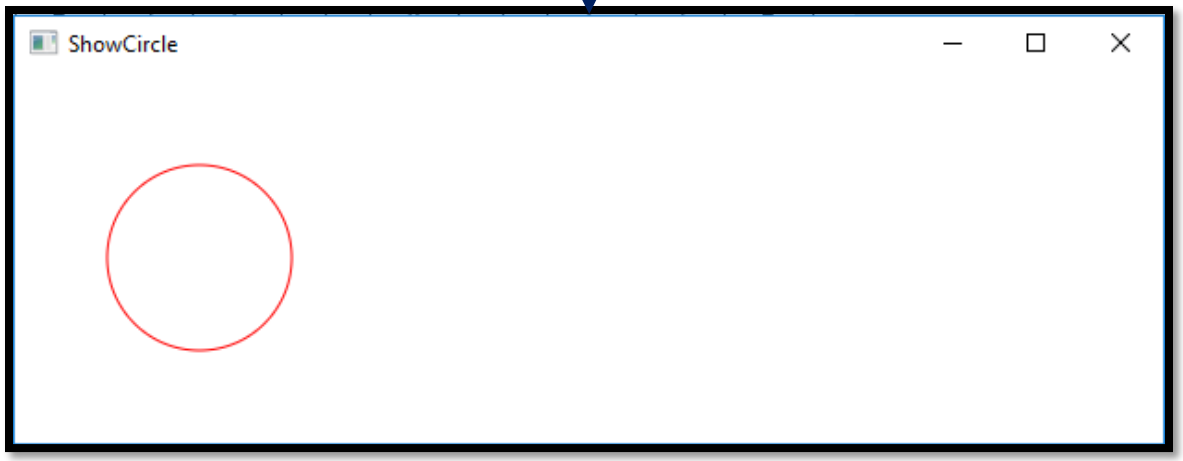


Property Binding

Property Binding

- In the previous example, the (x, y) location of the center of the circle was static – it will always be located at (100, 100).
- What if we want it to be centered in the pane, such that if we re-size the window, the circle will move to stay centered?
- In order to do so, the circle's center has to be *bound* to the pane's height and width, such that a change to the height or width will force a change to the x or y value of the circle's center.
- This is what *property binding* is all about





Property Binding

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

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

- When there's a change to the source, it gets automatically sent to the target.

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

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

The center of the `circle` will always remain in the middle of the `pane`

Property Binding

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

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
  }
}
```

x-coordinate of the circle center.

The target listens for changes in the source and updates itself when the source changes.

the binding syntax is:
target.bind(source);

Property Binding: Accessor, Mutator & Property Accessor

- Each binding property (e.g., `centerX`) in a JavaFX class (e.g., `Circle`) has the usual accessor (e.g., `getCenterX()`) and a mutator (e.g., `setCenterX(double)`)
- Each binding property also has a property accessor for returning the property itself. The name for this property accesor method should be the property name followed by the word `Property`

Syntax

```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

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

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

(a) `x` is a binding property

Example

```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

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

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

(b) `centerX` is binding property

Property Binding

JavaFX defines binding properties for primitive types and strings:

Type	Binding Property Type
double	DoubleProperty
float	FloatProperty
long	LongProperty
int	IntegerProperty
boolean	BooleanProperty
String	StringProperty

Example: Property Binding

LISTING 14.6 BindingDemo.java

```
1  import javafx.beans.property.DoubleProperty;
2  import javafx.beans.property.SimpleDoubleProperty;
3
4  public class BindingDemo {
5    public static void main(String[] args) {
6      DoubleProperty d1 = new SimpleDoubleProperty(1);
7      DoubleProperty d2 = new SimpleDoubleProperty(2);
8      d1.bind(d2);
9      System.out.println("d1 is " + d1.getValue()
10       + " and d2 is " + d2.getValue());
11     d2.setValue(70.2);
12     System.out.println("d1 is " + d1.getValue()
13       + " and d2 is " + d2.getValue());
14   }
15 }
```

```
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2
```

Property Binding

- The target listens for changes in the source and updates itself when the source changes
- Remember, the binding syntax is
  `target.bind(source);`
- Realize that with a setter, we specify a value; with binding, we specify the *property itself*, rather than the *value* of the property
- That's why we have to use the special methods `.add`, `.subtract`, `.multiply`, and `.divide`, rather than the numeric operators; the methods return *property objects*, rather than numeric values

Common Properties and Methods for Nodes

Common Properties and Methods for Nodes

- The Node class defines many properties and methods that are common to all nodes.
- Nodes share many common properties. This section introduces two such properties: **style** and **rotate.**
- JavaFX style properties are a lot like CSS (Cascading Style Sheets) use to specify styles in HTML (Web) pages.
- Thus, it's known as JavaFX CSS
- Each property begins with the prefix `-fx-`, and is of the form `styleName:value`, with multiple settings separated by semicolons.
- For example:

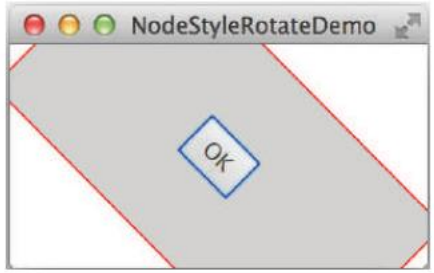
```
circle.setStyle("–fx–stroke: black; –fx–fill: red;");
```

- Is equivalent to:

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

Example: Node Style Rotate Demo

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

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

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

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



For example, the following code rotates a button 80 degrees: button.setRotate(80);

The `Color` Class

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.

The brighter() method returns a new Color with a larger red, green, and blue values

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.

A **color instance** can be constructed using the following constructor: **public** Color(**double** r, **double** g, **double** b, **double** opacity); in which **r**, **g**, and **b** specify a color by its red, green, and blue components with values in the **range from 0.0 (darkest shade) to 1.0 (lightest shade)**.

The **opacity value** defines the transparency of a color within the range from 0.0 (completely transparent) to 1.0 (completely opaque).

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

The Font Class

The Font Class

- We tend to think of "font" as being "Arial", "Times New Roman", "Calibri", etc., but those are font names (more precisely, they're typeface names)

- A Font is defined by its name, weight, posture, and size

The Font Class

- A Font describes font name, weight, and size.
- We typically use just "NORMAL" or "BOLD" for the FontWeights:

```
THIN
EXTRA_LIGHT
LIGHT
NORMAL
MEDIUM
SEMI_BOLD
BOLD
EXTRA_BOLD
BLACK
```

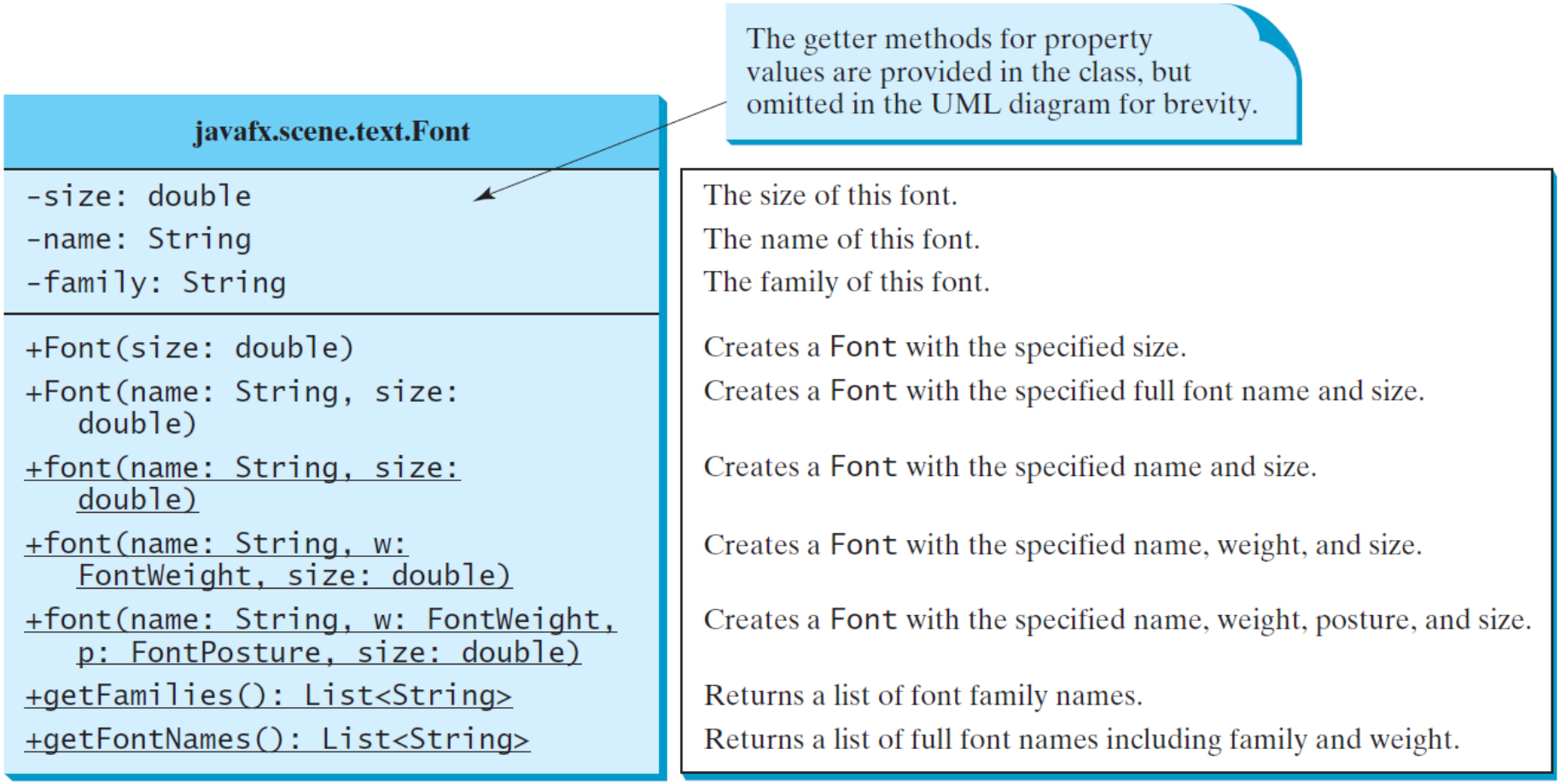
FontPosture, however, comes in exactly two flavors: REGULAR and ITALIC

You can get a listing of all of the font family names installed on the computer with .getFamilies()

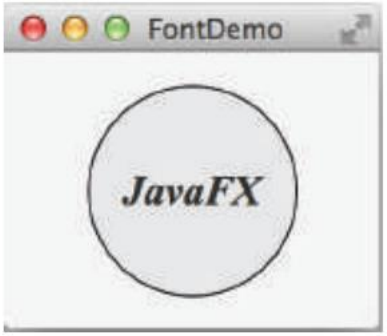
Example:

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

The Font Class



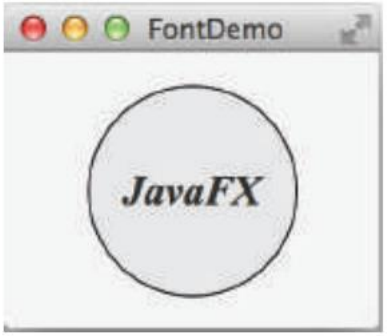
Example: Font Demo




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

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

Example: Font Demo

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

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

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

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

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



A label is on top of a circle displayed in the center of the scene.

- The program creates a StackPane , and adds a circle and a label to it

- These two statements can be combined using the following one statement: pane.getChildren().addAll(circle, label)

- **Why StackPane?**

The Image and ImageView Classes

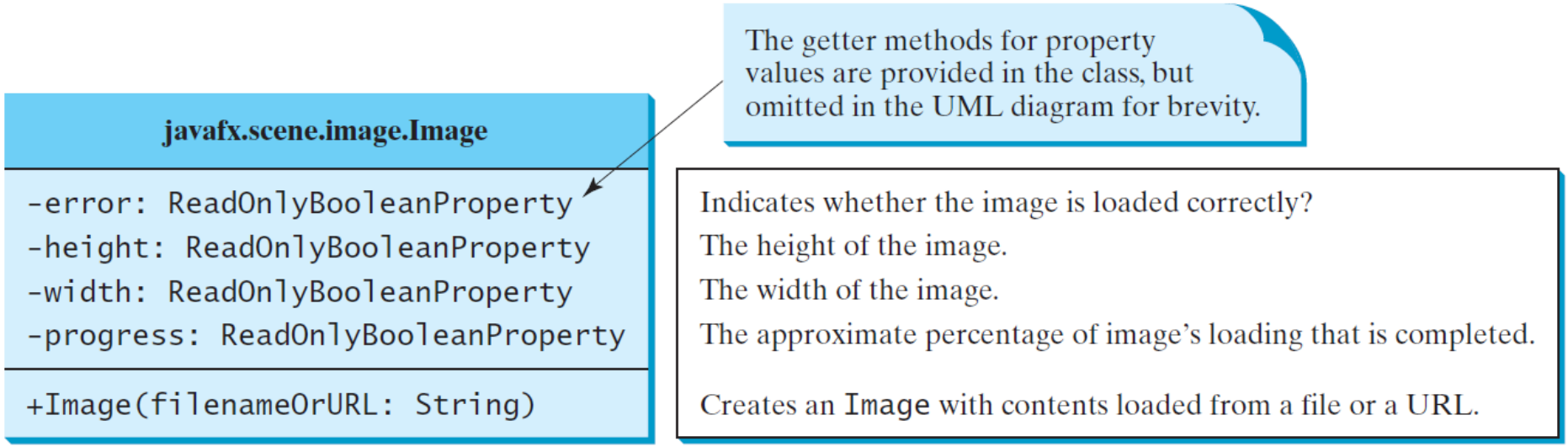
The Image & ImageView Classes

- We used the `File` class to hold information about a file, but not to actually read / write to it
  - For that we used a `Scanner` / `PrintWriter`, connected to the `File` object

- Similarly, the `Image` class is a container for an image, but can't be used to actually display an image
  - For that, we use the `ImageView` node (and attach it to a scene)

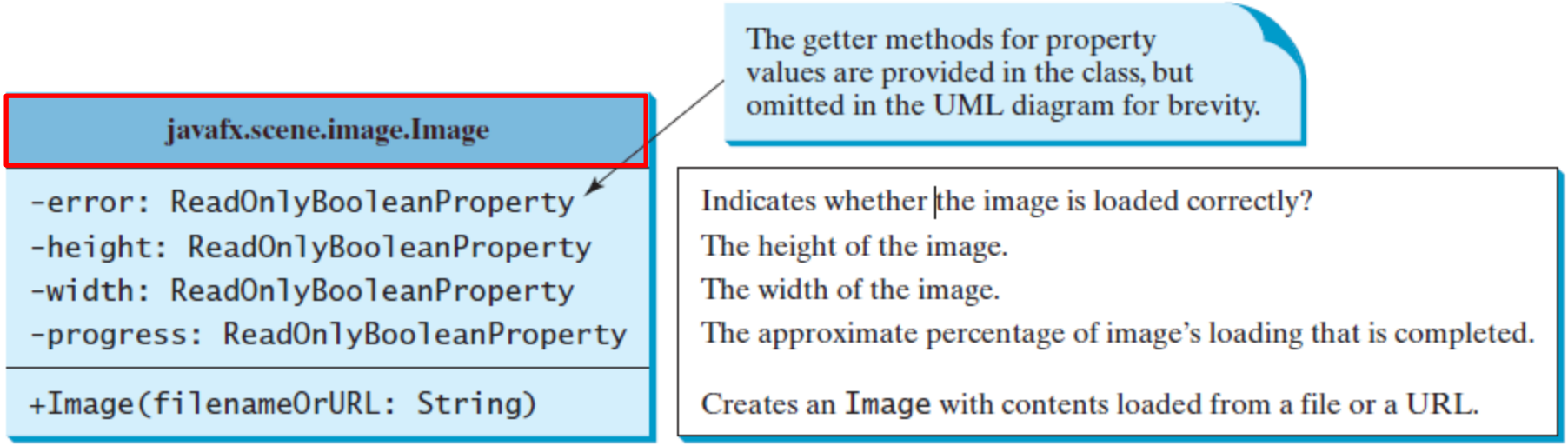
The Image and ImageView Classes

- The Image class represents a graphical image, and the ImageView class can be used to display an image.



The Image and ImageView Classes

- The `Image` class (the data types should be `ReadOnlyDoubleProperty` for all except `error`):



- We construct an Image from a filename (or a URL), and then we can give the Image to an ImageView object to actually display it

The ImageView Class

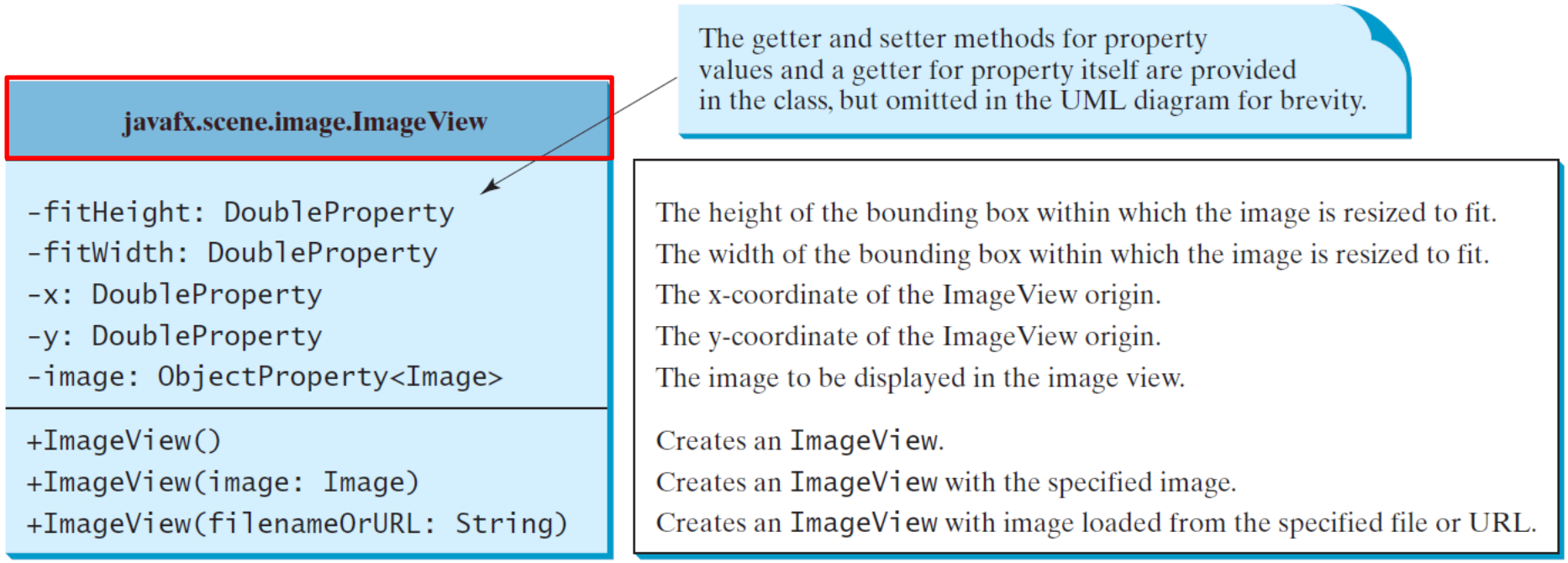
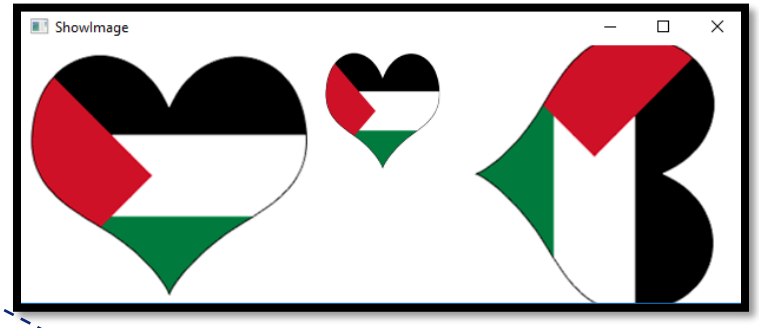


image and the ImageView class can be used to display an image.

Example:
The Image & ImageView Classes

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

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

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

    ImageView imageView3 = new ImageView(image);
    imageView3.setRotate(90);
    pane.getChildren().add(imageView3);

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

new Insets (top, right, down, left)



set image view properties

rotate an image view

Notes: The Image & ImageView Classes

- The HBox is a pane that handles placement of multiple nodes for us automatically.
- As we add nodes to the HBox, they are automatically added in a row (horizontally)
- setRotate is a method in the Node class, so all nodes can be rotated.
- If you use the URL-based constructor for Image, it must include "http://"
- Java assumes the image is located in the same directory as the .class file. If it's located elsewhere, you must use either a full path or a relative path to specify where

Layout Panes

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.

Pane: Base class for layout panes. Use its getChildren() method to return the list of nodes on the pane (or add to that list) Provides no particular layout capabilities – it's a "blank canvas" typically used to draw shapes on

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.

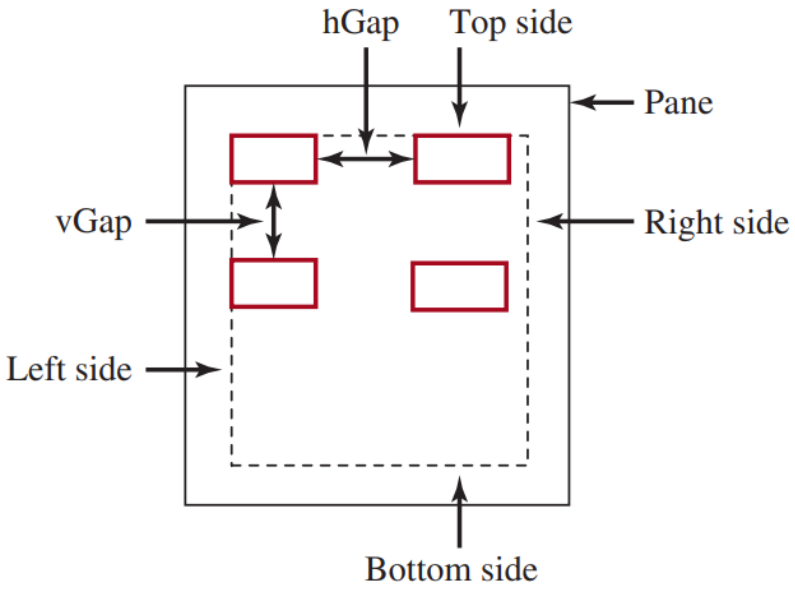
Creates a `FlowPane` with a specified orientation.

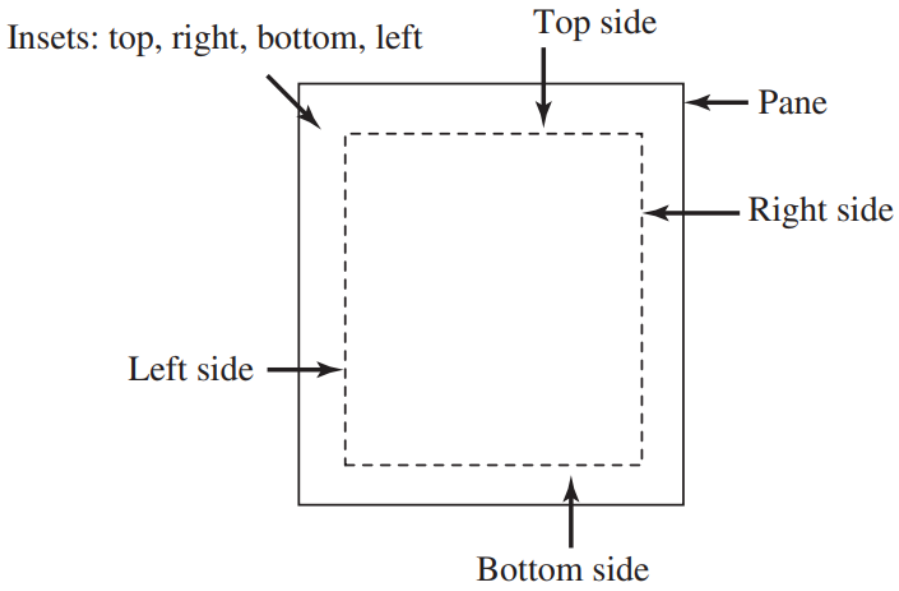
Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

- FlowPane arranges the nodes in the pane **horizontally from left to right** or **vertically from top to bottom in the order in which they were added**. When one row or one column is filled, a new row or column is started.
  - The `FlowPane` can be set up to work in "reading order" (sequential rows left-to-right), by using `Orientation.HORIZONTAL` or in sequential top-to-bottom columns (`Orientation.VERTICAL`))
- You can also specify the **gap between nodes (in pixels)**

FlowPane

- You can specify the way the nodes are placed horizontally or vertically using one of two constants: **Orientation.HORIZONTAL** or **Orientation.VERTICAL**. You can also specify the gap between the nodes in pixels.



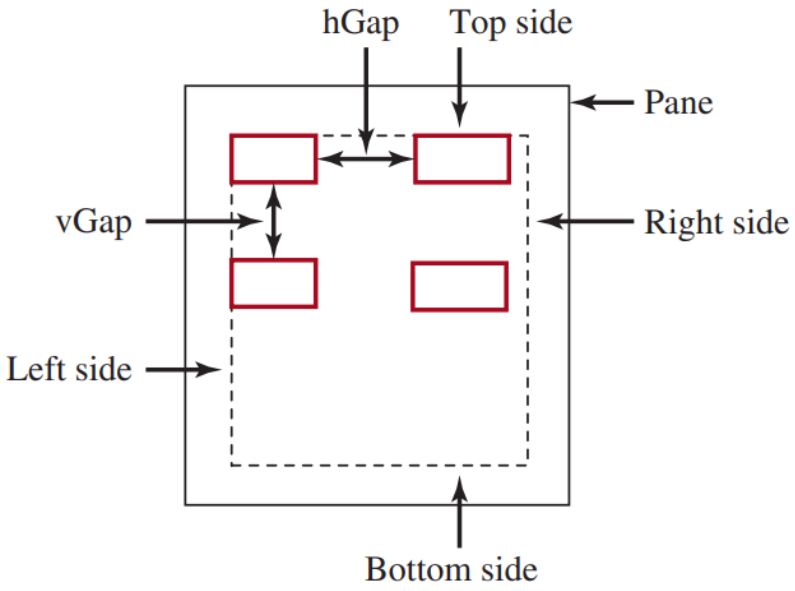


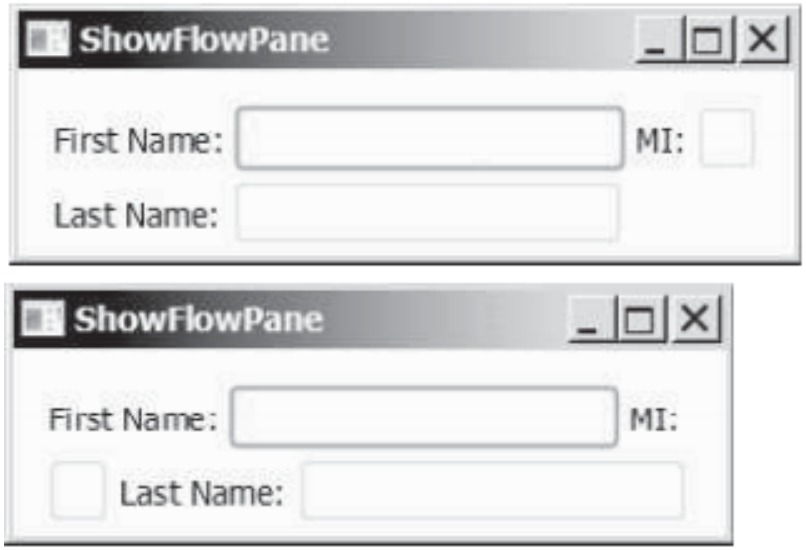
You can specify `hGap` and `vGap` between the nodes in a `FlowLPane`.

- To add nodes : add(node) or addAll(node1, node2, ...) method.
- To remove nodes:remove(node) or use the removeAll() method to remove all nodes from the pane

FlowPane

- You can specify the way the nodes are placed horizontally or vertically using one of two constants: **Orientation.HORIZONTAL** or **Orientation.VERTICAL**. You can also specify the gap between the nodes in pixels.



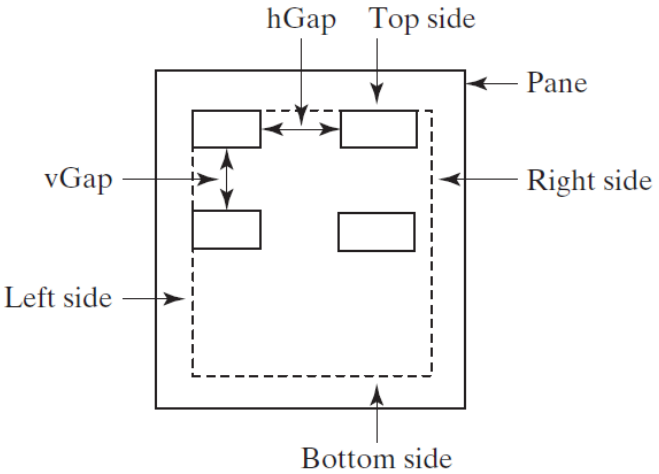


You can specify **hGap** and **vGap** between the nodes in a **FlowLPane**.

- To add nodes : add(node) or addAll(node1, node2, ...) method.
- To remove nodes:remove(node) or use the removeAll() method to remove all nodes from the pane

Example: The FlowPane

```
public void start(Stage primaryStage) // From Listing 14.10 (p. 553)
{
    // Create a pane and set its properties
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);
```




```
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Setting the padding with an `Insets` object gives us a margin *inside* the pane. Just as angles are always clockwise, the `Insets` are specified in clockwise order from the top, so this pane will have an 11-pixel gap between the top of the pane and the first row, a 12-pixel gap between the right-most node and the right side of the pane, 13 pixels at the bottom, and 14 pixels on the left side.

The `Hgap` and `Vgap` properties specify the gap between elements on the same row, or between rows

Resizing the window "re-flows" the nodes, just as changing the page size in Word re-flows your text.

Example: The FlowPane

This example introduces two new nodes: `Label` (which just lets us display text on a pane), and `TextField` (which provides a box into which the user can type text).

`TextField` nodes typically have a corresponding `Label`, so the user can tell what's supposed to go IN the `TextField`

```
// 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
```

We can `.add()` individual nodes to a pane, or we can `.addAll()` to add a *list* of nodes, as is done here.

We add a `Label` of "First Name:", and then a `TextField` into which the user can type their first name, and then another `Label` for "MI:" ("Middle Initial").

Example: The FlowPane

```
public void start(Stage primaryStage)
{
   // Create a pane and set its properties
   FlowPane pane = new FlowPane();
   pane.setPadding(new Insets(11, 12, 13, 14));
   pane.setHgap(5);
   pane.setVgap(5);

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

   // Create a scene and place it in the stage
```

Next, we create another `TextField` for the Middle Initial, and set its preferred column count to 1 (if it's only going to hold *an* initial, why make a "wide" box to hold it?)

Note: the `TextField`'s variable is prefixed with "`tf`". Node variables are typically prefixed with an abbreviation of its type, so we can tell from looking at the variable what *kind* of variable it is

Example: The FlowPane

```
public void start(Stage primaryStage)
{
   // Create a pane and set its properties
   FlowPane pane = new FlowPane();
   pane.setPadding(new Insets(11, 12, 13, 14));
   pane.setHgap(5);
   pane.setVgap(5);

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

   // Create a scene and place it in the stage
```

Finally, we go back to the task of adding the (narrow) Middle Initial `TextField`, plus a `Label` and a `TextField` for the *last* name to the pane.

Now we have `Label` / `TextField` pairs for First Name, Middle Initial (a one-character-wide `TextField`), and Last Name

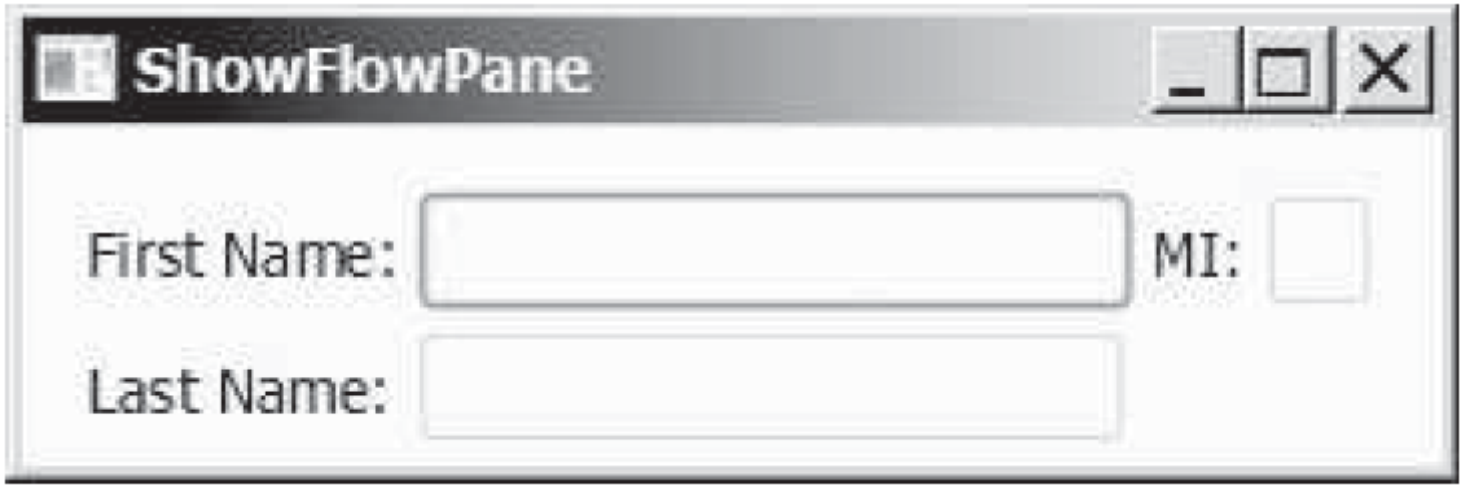
Example: The FlowPane

- This is the stage we have built:

- Because we went in "reading order", and added the nodes in the order we did, we got this – the "MI:" label *is* next to the `tfMi TextField` *in reading order*, but this isn't visually appealing.

Example: The FlowPane

- If we make the scene wider (enlarge the window)



- The elements “re-flow”, and now the MI TextField fits on the same row as its Label

GridPane

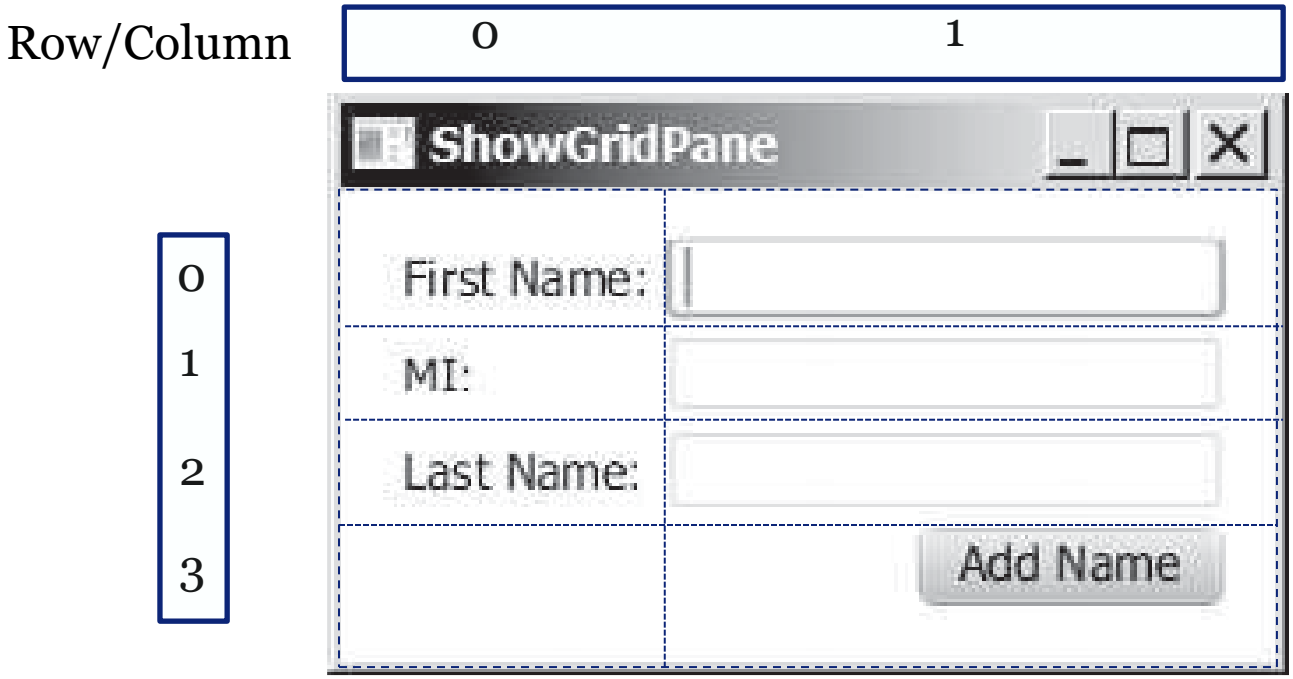
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane

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

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

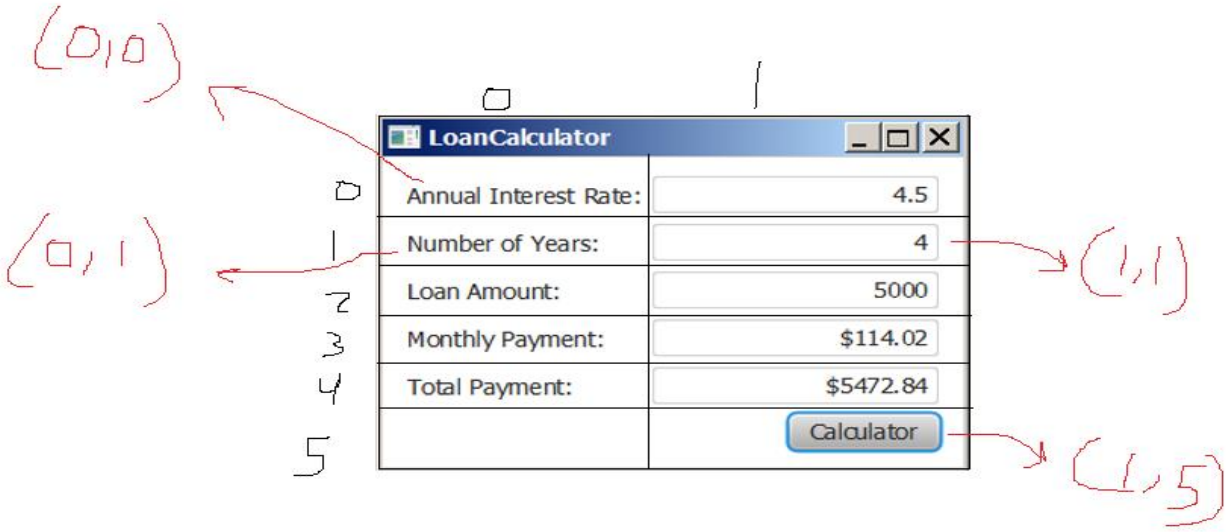
Example: GridPane



Example: GridPane

- A few notes:
  - The "Add" button is right-aligned within its cell //GridPane.setHalignment(btAdd, HPos.RIGHT)
  - Alignment is set to the center position, which causes the nodes to be placed in the center of the grid pane. //pane.setAlignment(Pos.CENTER);
  - The labels get the default horizontal alignment of "left"
  - We specify the column first (backwards from arrays) //Example: A button is placed in column 1 and row 3
  - Not every cell needs to be filled// see the last row, we have only button
  - Elements can be moved from one cell to another

Example: GridPane



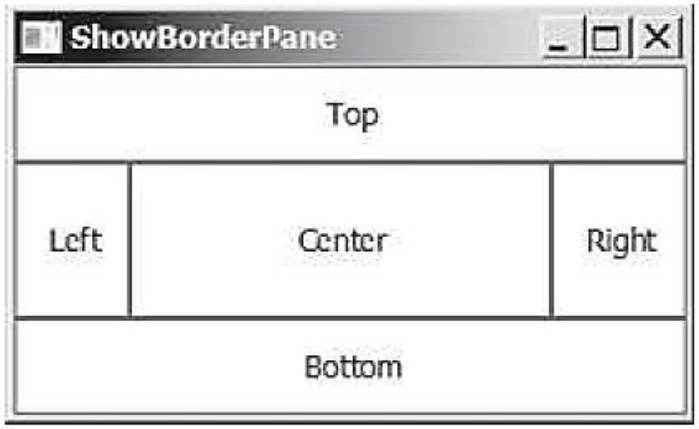

```
pane.add(new Label("First Name:"), 0, 0);
```

column

row

The BorderPane

- The `BorderPane` divides the pane into **five regions**
- Note that a `Pane` is also a `Node`, so a `Pane` can contain another `Pane`
- If a region is empty, it's simply not shown
- We can clear a region with `set<region>(null)`



BorderPane

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

javafx.scene.layout.BorderPane

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

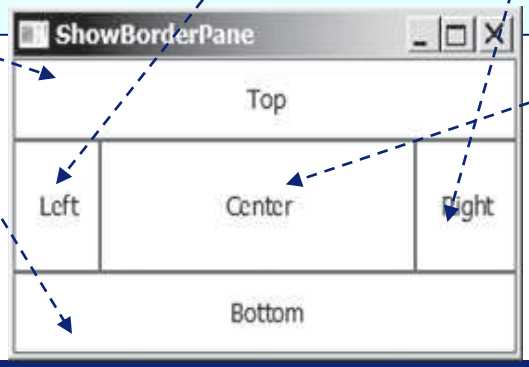
```
+BorderPane()
+setAlignment(child: Node, pos: Pos)
```

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`).

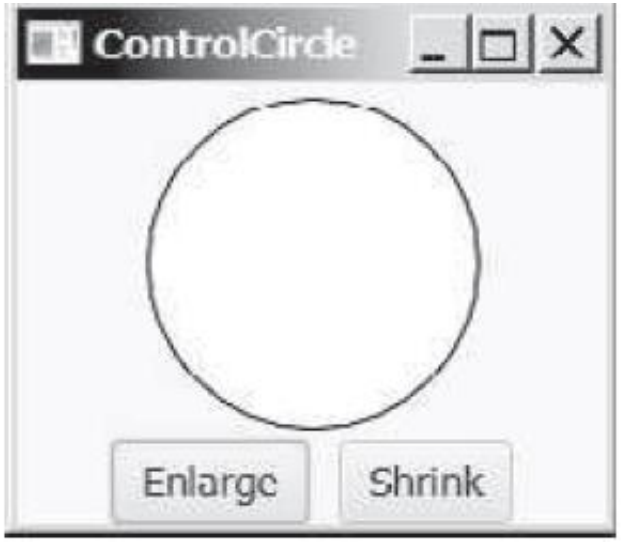
Creates a `BorderPane`.

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

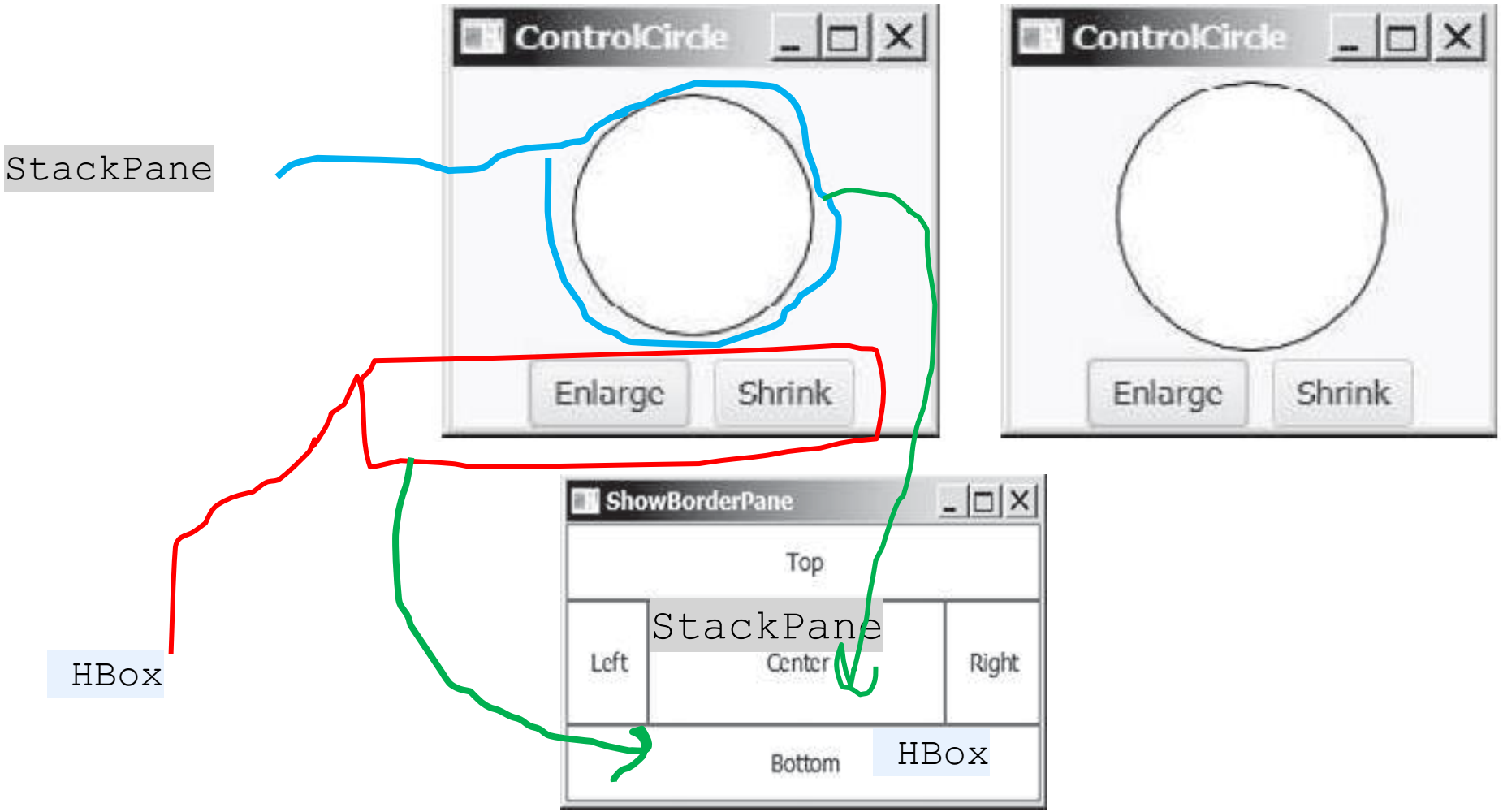
A **BorderPane** can place nodes in five regions: top, bottom, left, right, and center, using the **setTop(node), setBottom(node), setLeft(node), setRight(node), and setCenter(node) methods**.



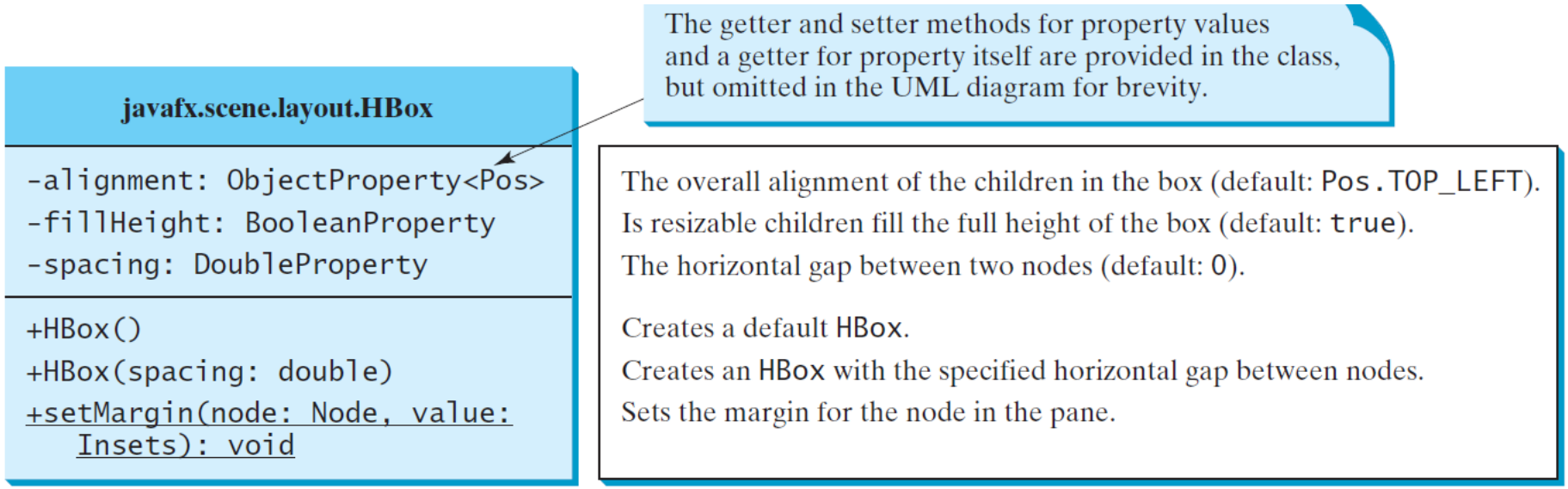
Example: BorderPane



Example: BorderPane

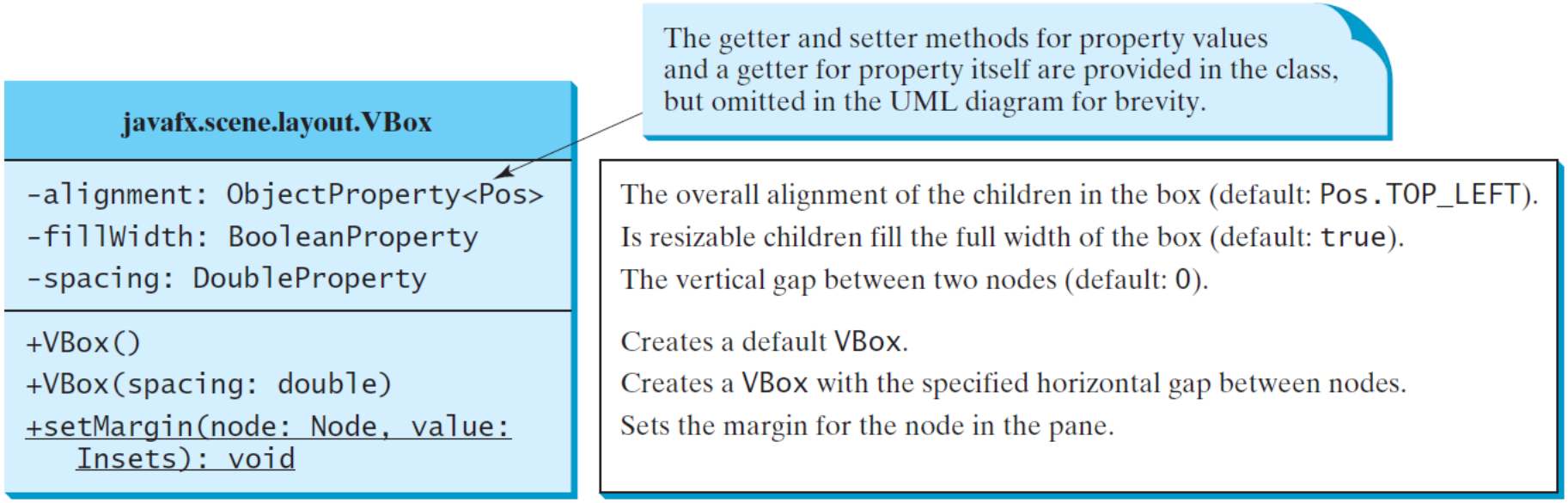


HBox



The HBox is a pane that handles placement of multiple nodes for us automatically.
As we add nodes to the HBox, **they are automatically added in a row (horizontally)**

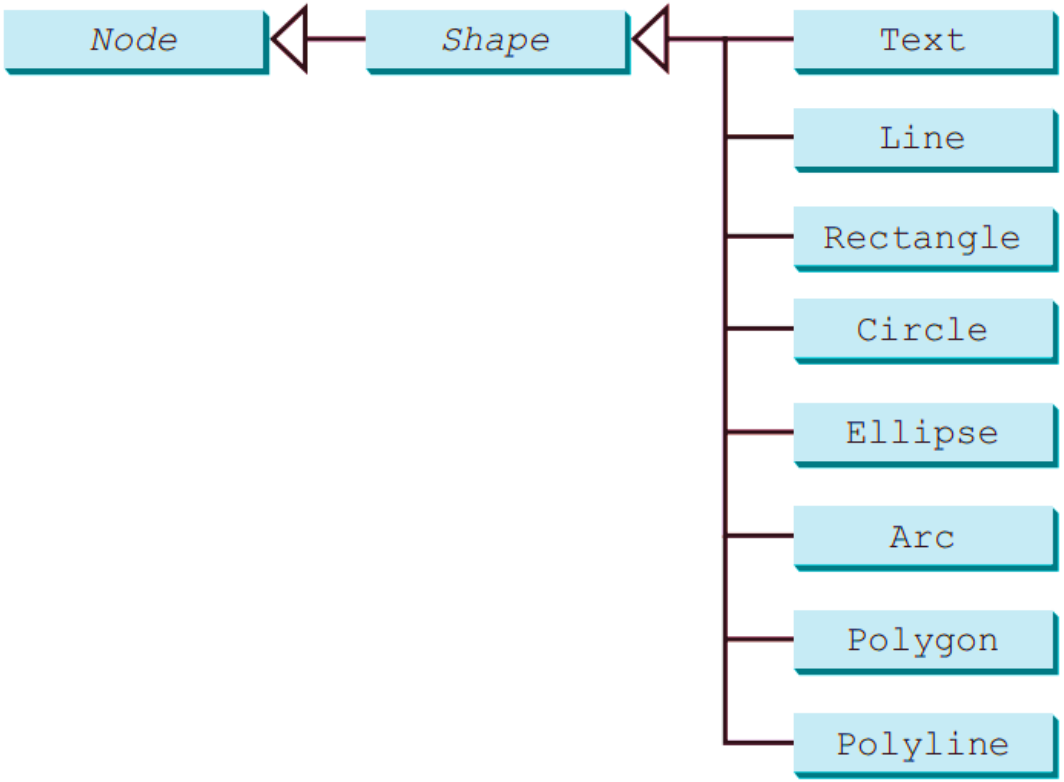
VBox


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.

Shapes

Shapes

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



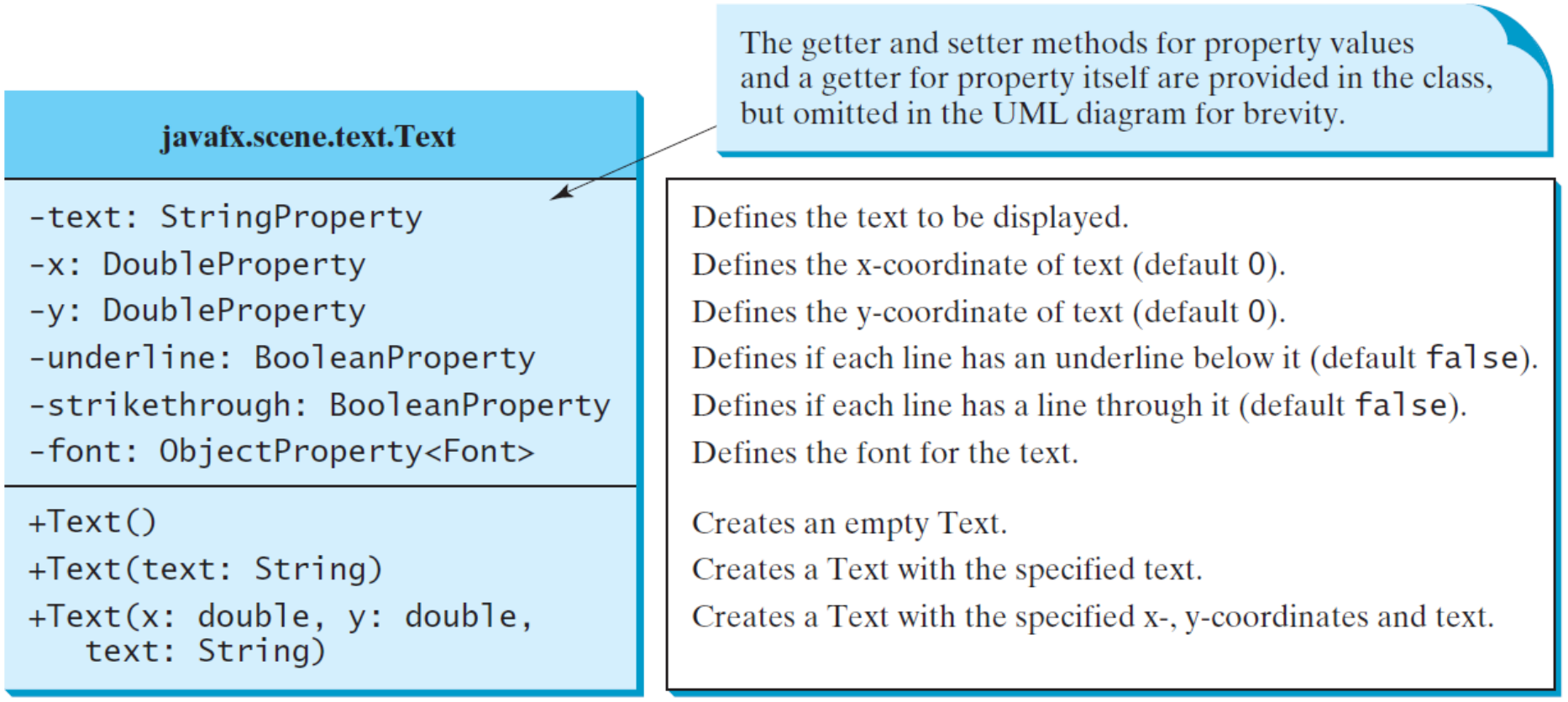
A shape is a node. The **Shape** class is the root of all shape classes.

Shapes

- All of the shapes have some common properties:
  - fill (color)
  - stroke (line color)
  - strokeWidth (how heavy the stroke line is)

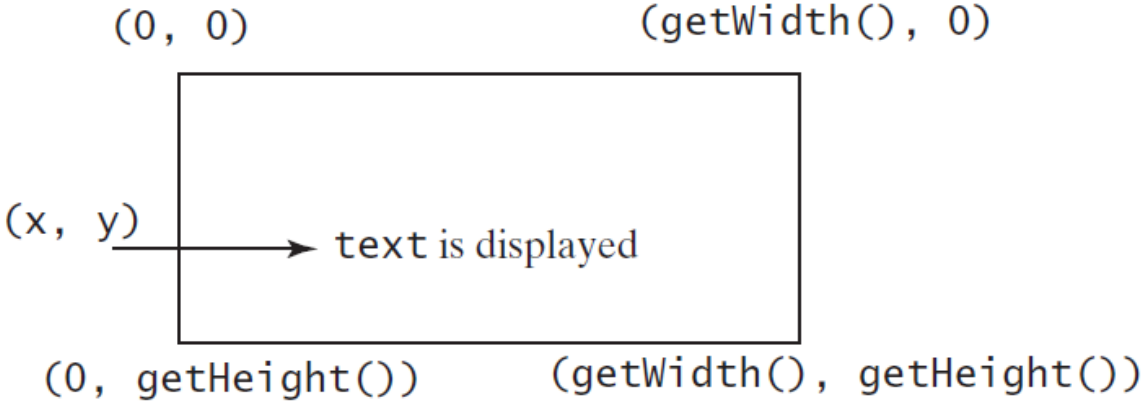
86

Text


javafx.scene.text.Text
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
+Text()
+Text(text: String)
+Text(x: double, y: double, text: String)
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.

Text Example

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



(a) Text(x, y, text)

(b) *Three* Text *objects are displayed*

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.

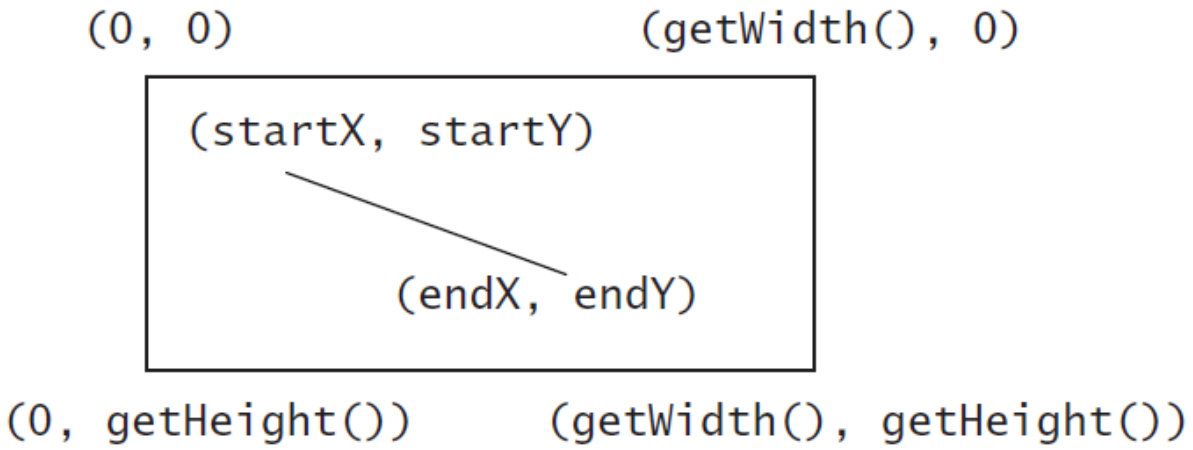
javafx.scene.shape.Line

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

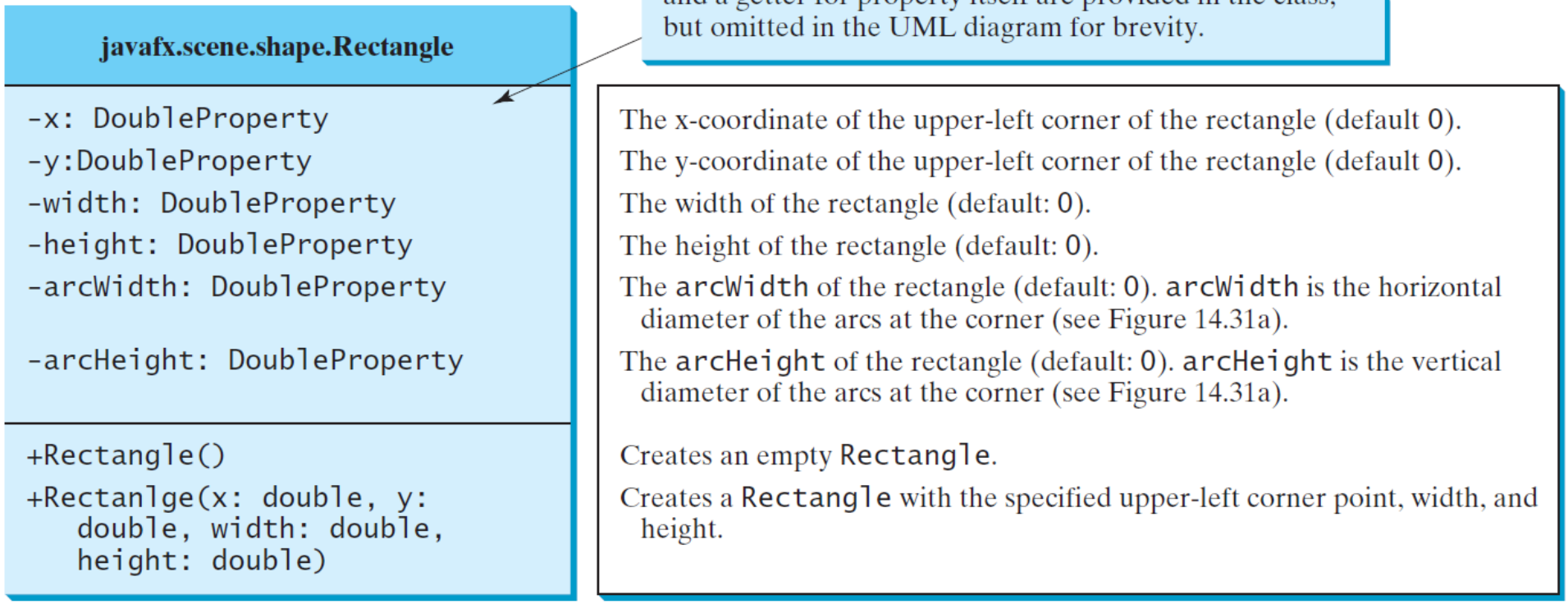
```
+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

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.

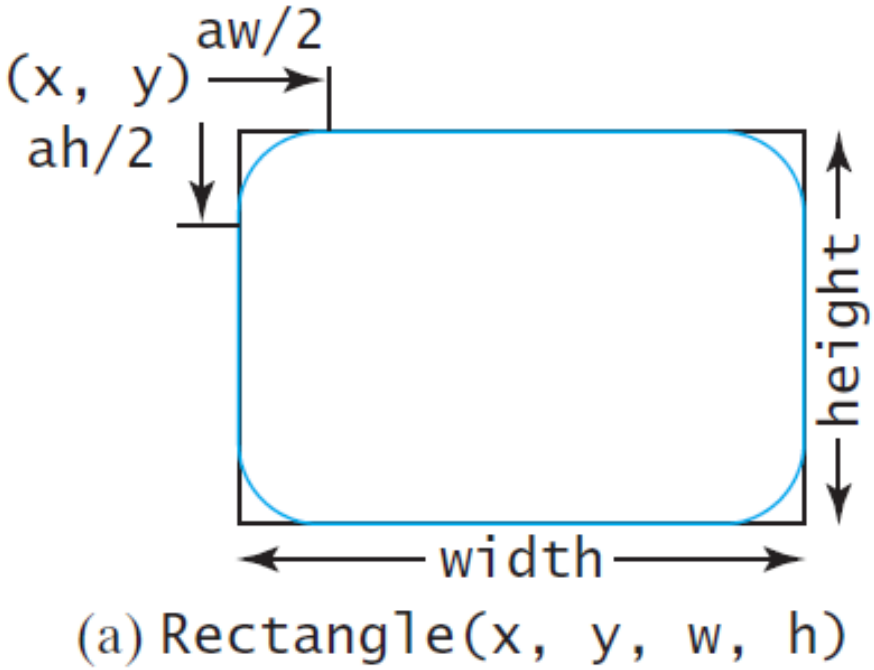


Rectangle


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 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.

Rectangle Example

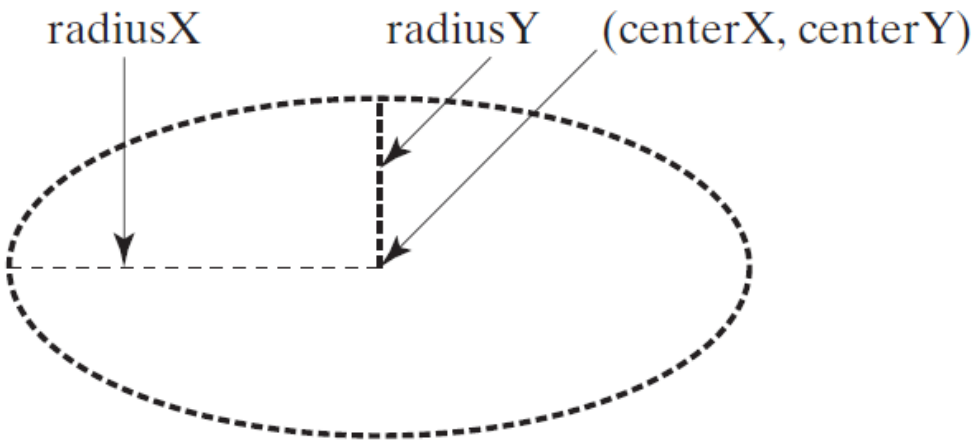
- The `Rectangle` shape is specified by its top-left corner (`x`, `y`) and its `width` and `height`.
- Optionally, we can specify the `arcWidth` and `arcHeight` (in pixels) of the corners, to create rounded corners (arc measurements of 0 make squared-off corners)



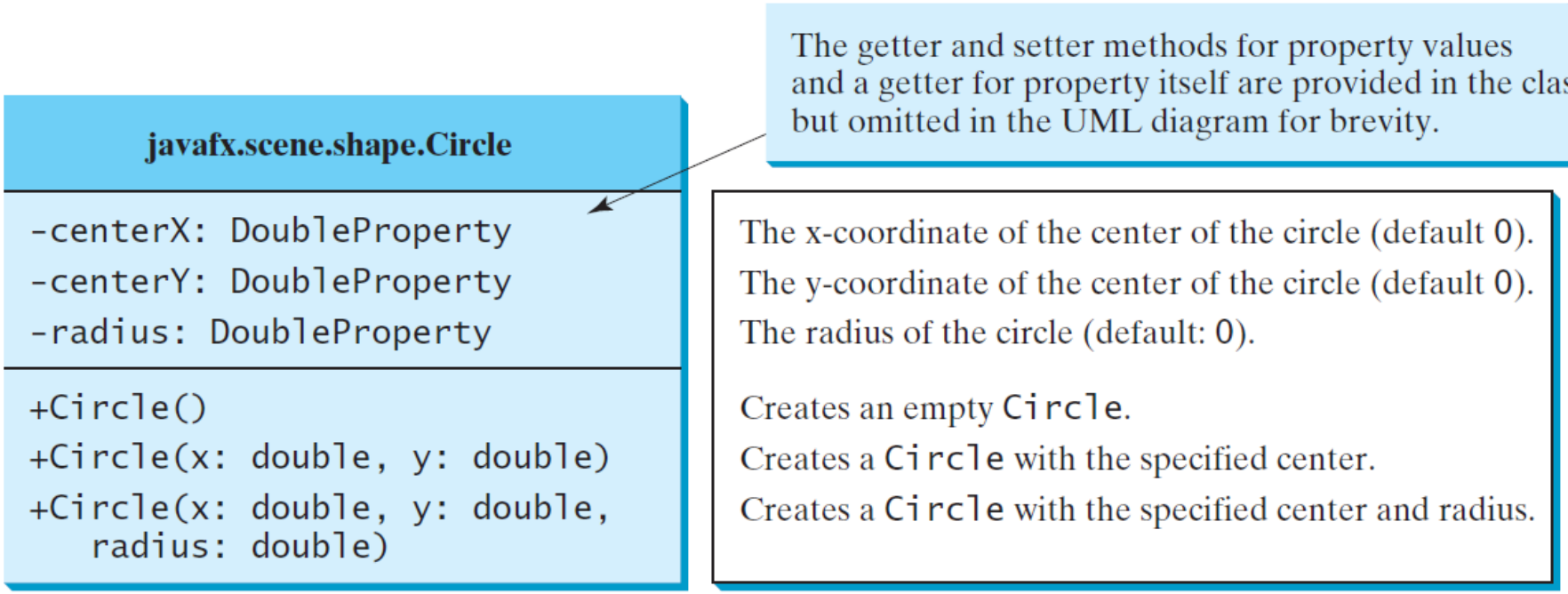
(a) Rectangle(x, y, w, h)

Circle and Ellipse

- The Circle and Ellipse shapes are very similar.
- Both are specified by their center point (centerX, centerY)
- The Circle has a single radius
- The Ellipse has *two* radii (radiusX & radiusY)



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

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.

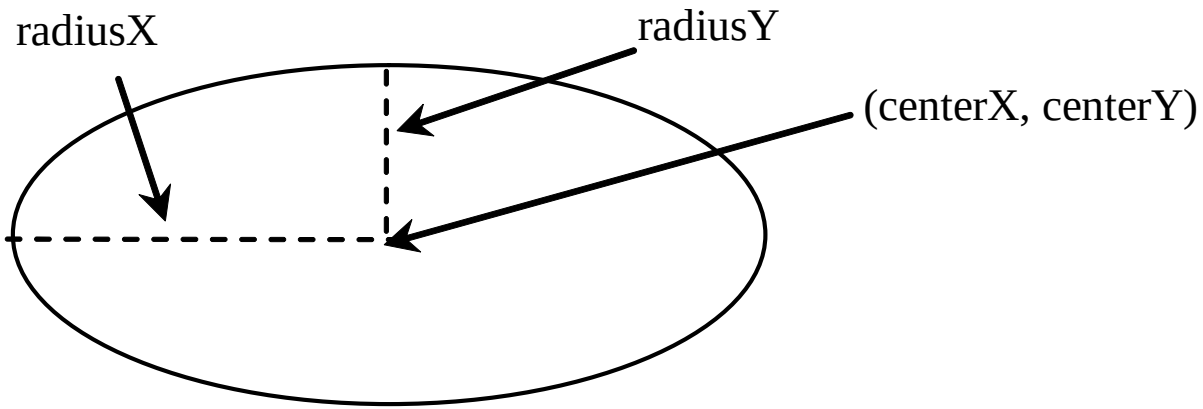
javafx.scene.shape.Ellipse

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

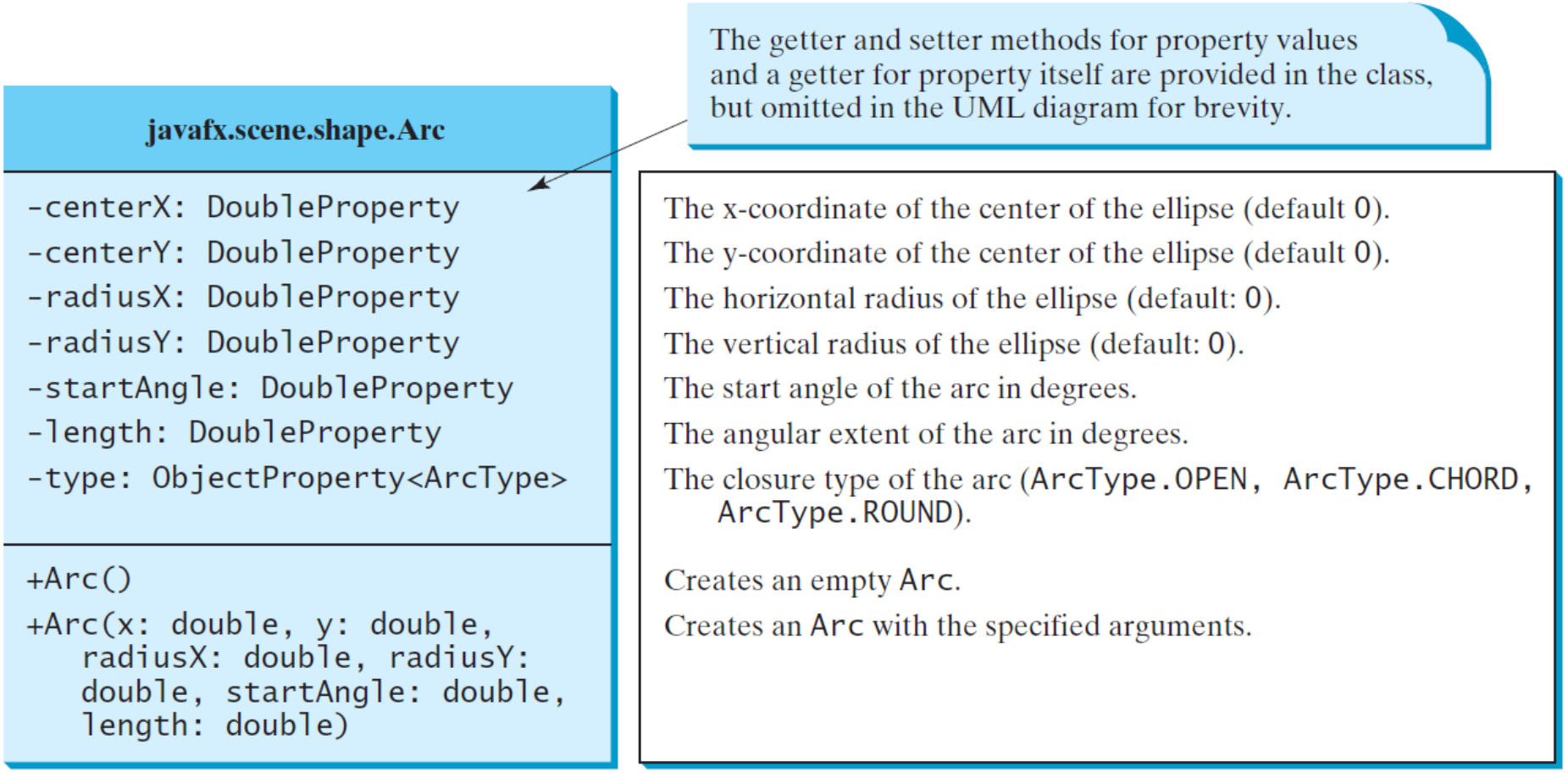
```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

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).

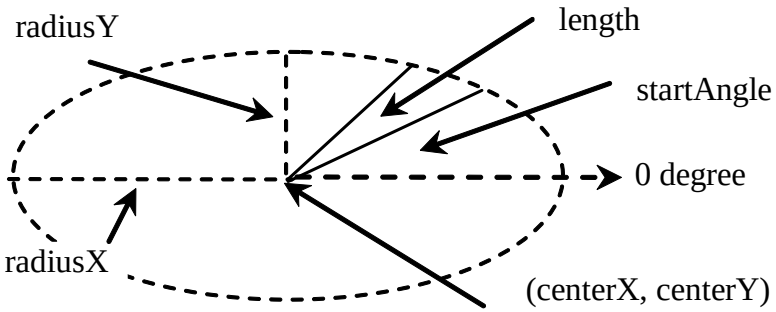
Creates an empty `Ellipse`.
Creates an `Ellipse` with the specified center.
Creates an `Ellipse` with the specified center and radiuses.

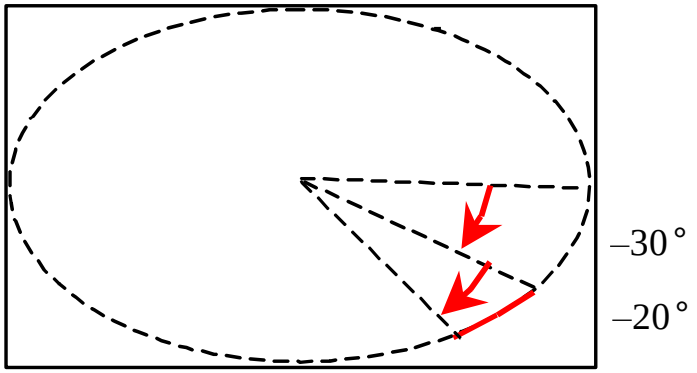


Arc

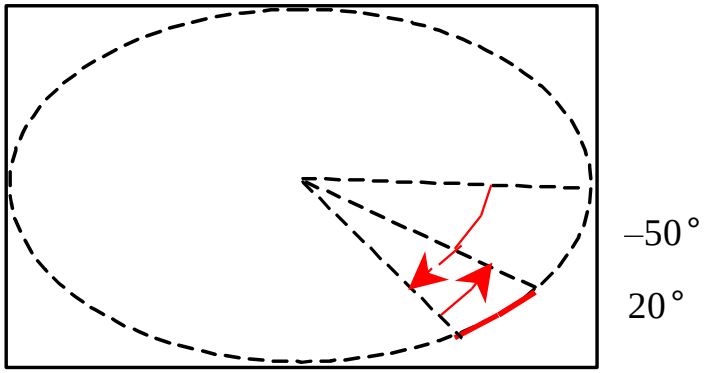

javafx.scene.shape.Arc
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
+Arc()
+Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: double)
The 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).
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.

Arc Examples





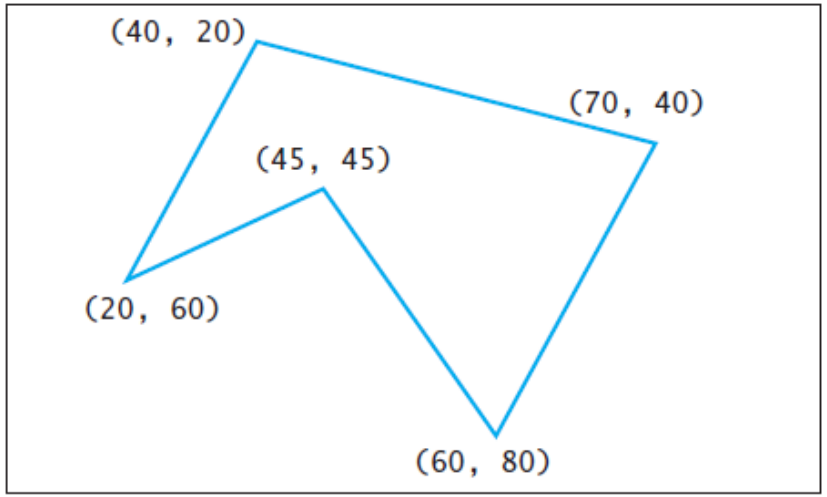
(a) Negative starting angle –30° and negative spanning angle –20°



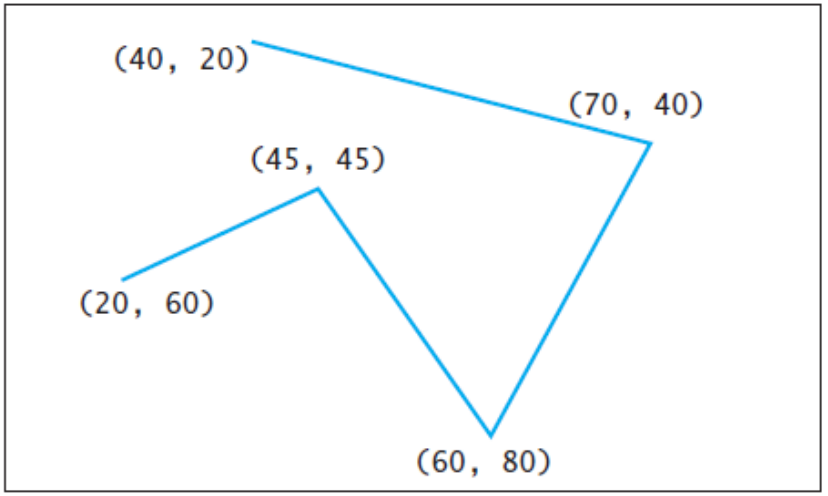
(b) Negative starting angle –50° and positive spanning angle 20°

Polygon and Polyline

- The `Polygon` and `Polyline` shapes are identical, except that the `Polyline` isn't closed
- Both are specified by a list of (x, y) pairs

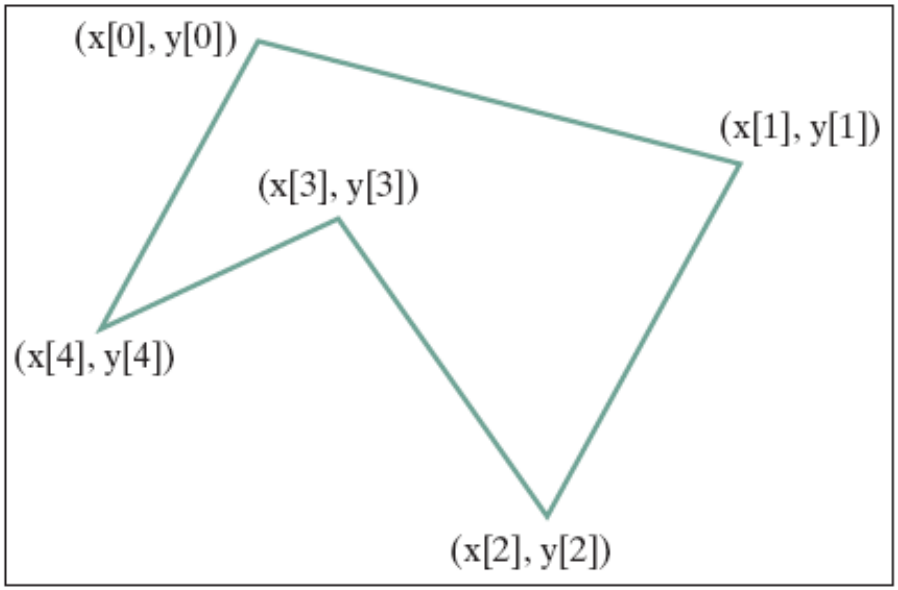


(a) Polygon

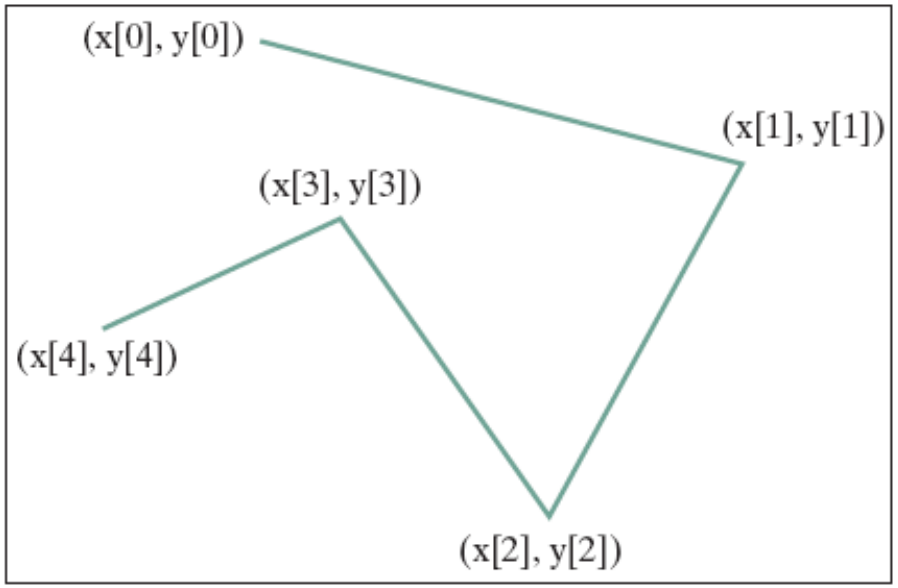


(b) Polyline

Polygon and Polyline

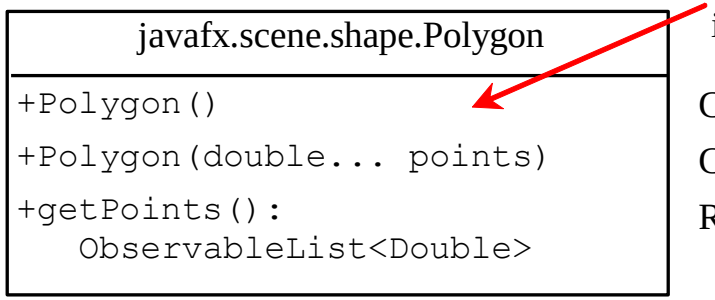


(a) Polygon



(b) Polyline

Polygon



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.

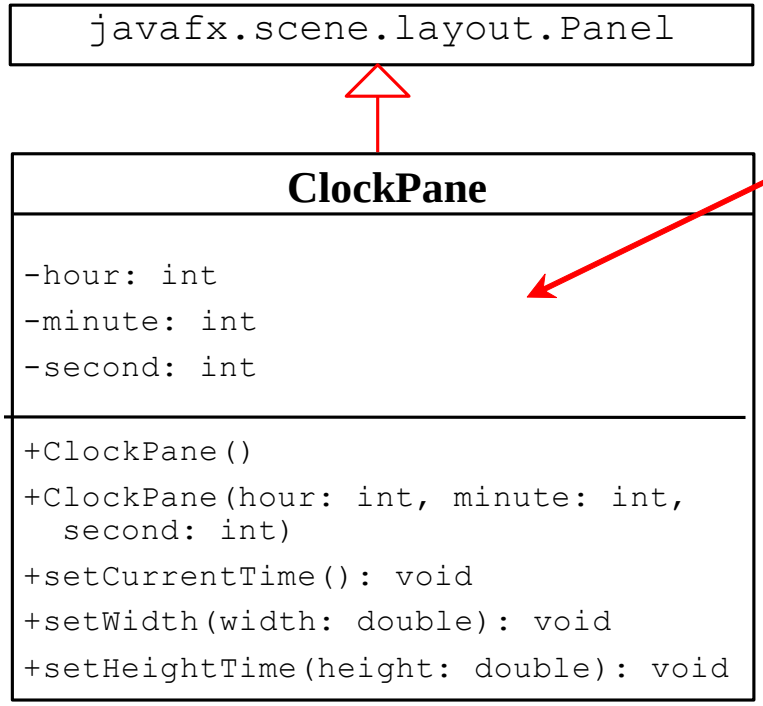
Creates an empty polygon.

Creates a polygon with the given points.

Returns a list of double values as x- and y-coordinates of the points.

Case Study: The ClockPane Class

This case study develops a class that displays a clock on a pane.



The `getter` and `setter` methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The hour in the clock.

The minute in the clock.

The second in the clock.

Constructs a default clock for the current time.

Constructs a clock with the specified time.

Sets hour, minute, and second for current time.
Sets clock pane's width and repaint the clock,
Sets clock pane's height and repaint the clock,

Use the ClockPane Class

GUI development and JavaFX Basics