Chapter 14
JavaFX Basics

Motivations

- JavaFX is a new framework for developing Java GUI programs. The JavaFX API is an excellent example of how the object-oriented principle is applied.
- This chapter serves two purposes. First, it presents the basics of JavaFX programming. Second, it uses JavaFX to demonstrate OOP.
- Specifically, this chapter introduces the framework of JavaFX and discusses JavaFX GUI components and their relationships.

Objectives(1)

- To distinguish between JavaFX, Swing, and AWT (§14.2).
- To write a simple JavaFX program and understand the relationship among stages, scenes, and nodes (§14.3).
- To create user interfaces using panes, groups, UI controls, and shapes (§14.4).
- To update property values automatically through property binding (§14.5).
- To use the common properties style and rotate for nodes (§14.6).
- To create colors using the Color class (§14.7).

3

Objectives(2)

- To create fonts using the Font class (§14.8).
- To create images using the Image class and to create image views using the ImageView class (§14.9).
- To layout nodes using Pane, StackPane, FlowPane, GridPane, BorderPane, HBox, and VBox (§14.10).
- To display text using the Text class and create shapes using Line, Circle, Rectangle, Ellipse, Arc, Polygon, and Polyline (§14.11).
- To develop the reusable GUI components ClockPane for displaying an analog clock (§14.12).

4

14.2 JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications.
- When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT). AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. 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 known as JavaFX.

5

JavaFX

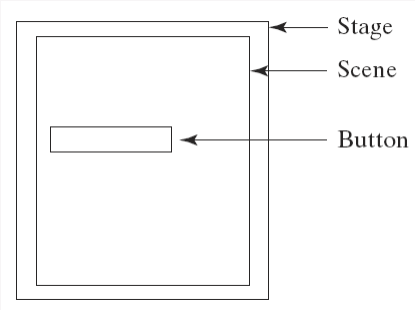
- JavaFX incorporates modern GUI technologies to enable you to develop rich Internet applications. A rich Internet application (RIA) is a Web application designed to deliver the same features and functions normally associated with deskop applications.
- A JavaFX application can run seemlessly on a desktop and from a Web browser.
- Additionally, JavaFX provides a multi-touch support for touch-enabled devices such as tablets and smart phones.
- JavaFX has a built-in 2D, 3D, animation support, video and audio playback, and runs as a stand-alone application or from a browser.

Why JavaFX?

- This book teaches Java GUI programming using JavaFX for 3 reasons.
  - First, JavaFX is much simpler to learn and use for new Java programmers.
  - Second, JavaFX is a better pedagogical tool for demonstrating object-oriented programming than Swing.
  - Third, Swing is essentially dead, because it will not receive any further enhancement.
- JavaFX is the new GUI tool for developing cross-platform-rich Internet applications on desktop computers, on hand-held devices, and on the Web.

14.3 The Basic Structure of a JavaFX Program

- *The abstract **javafx.application.Application** class defines the essential framework for writing JavaFX programs.*
- Application
- Override the start(Stage) method
- Stage, Scene, and Nodes
  - Stage is a window for displaying a scene that contains nodes.



MyJavaFX | Run

MultipleStageDemo | Run

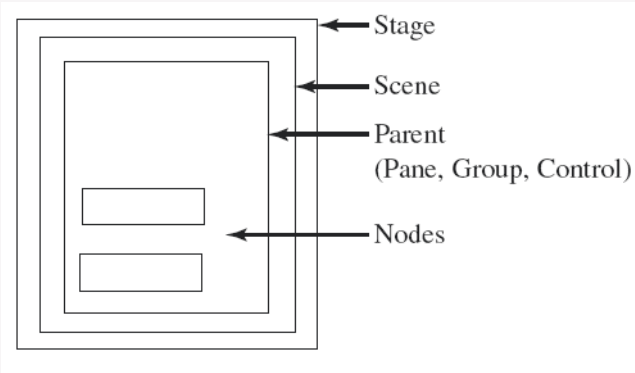
14.4 Panes, Groups, UI Controls, and Shapes

- *A pane* is a container used to hold nodes.
- A *node is a visual component such as a shape, an image view, a UI control,* or a pane.
- A *shape refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.*
- A *UI control refers to a label, button, check box, radio button, text field, text area, etc.*
- A *group* is a container that groups a collection of nodes. You can apply transformations or effects to a group, which automatically apply to all the children in the group.
- *A scene* can be displayed in a stage. *You place nodes inside a pane and then place the pane into a scene.*

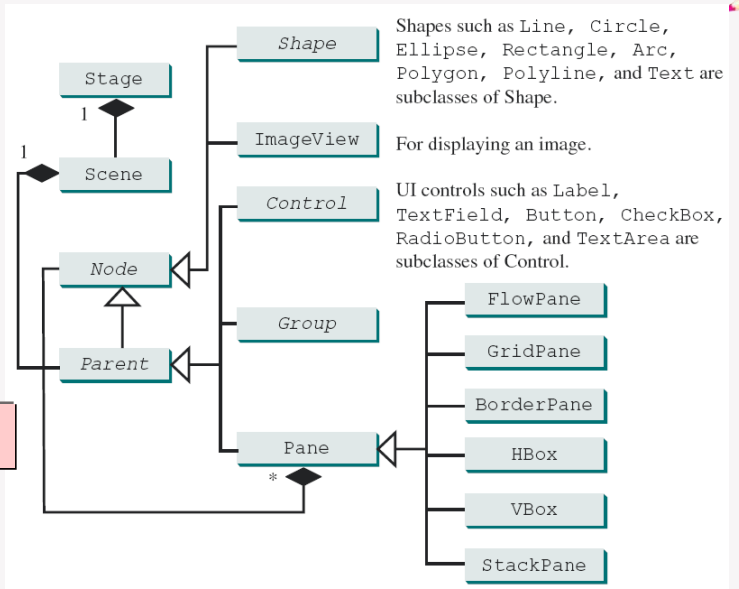
9

Panes, Groups, UI Controls, and Shapes

- Panes and groups are used to hold nodes
- Nodes can be shapes, image views, UI controls, groups, and panes.



The Relationship among Stage, Scene, Node, Control, and Pane

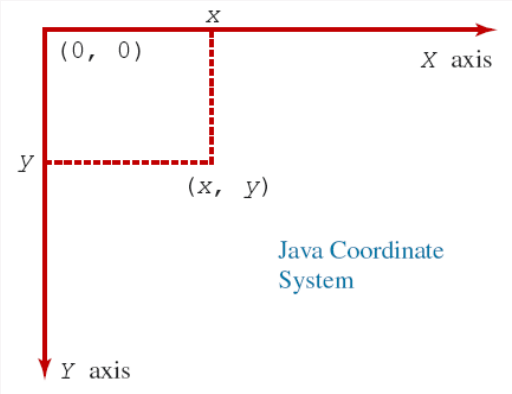


ButtonInPane

Run

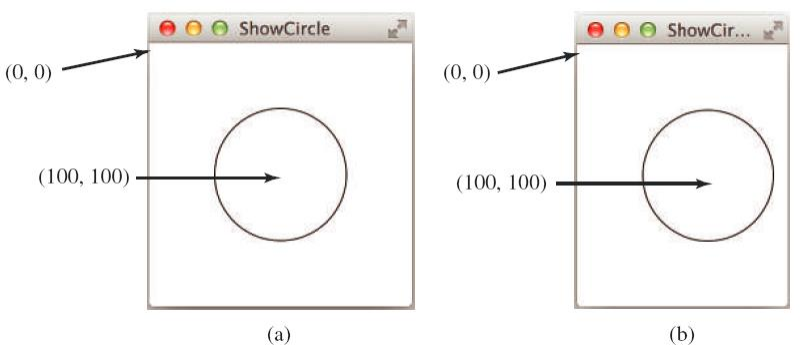
Display a Shape

- The Java coordinate system is measured in *pixels*, with **(0, 0)** at its upper-left corner.



Display a Shape

- This example displays a circle in the center of the pane.
- A circle is displayed in the center of the scene.
- The circle is not centered after the window is resized.



ShowCircle

Run

14.5 Property Binding

- JavaFX introduces a new concept called ***property binding*** *that enables a target object to be* bound to a *source object.*
- If the value in the source object changes, the target object is also changed automatically.
- The target object is called a *binding object or a binding property and* the source object is called a *bindable object or observable object.*

ShowCircleCentered

Run

Binding Properties

- The bind method is defined in the **javafx.beans.property.Property** interface. A target binds with a source using the **bind** method as follows:

```
target.bind(source);
```

- A binding property is an instance of **javafx.beans.property.Property.**
- A source object is an instance of the **javafx.beans.value.ObservableValue** interface. An **ObservableValue** is an entity that wraps a value and allows to observe the value for changes.
- JavaFX defines binding properties for primitive types and strings.
  - DoubleProperty/FloatProperty/LongProperty/IntegerProperty/BooleanProperty/StringProperty

Binding Property: getter, setter, and property getter

- By convention, each binding property in a JavaFX class has a getter and setter method for returning and setting the property's value. It also has a getter method for returning the property itself. The naming convention for this method is the property name followed by the word Property.
- We call the getCenterX() method as the **value getter** method, the setCenterX(double) method as the **value setter** method, and centerXProperty() as the **property getter** method.
  - Note that getCenterX() returns a double value and centerXProperty() returns an object of the DoubleProperty type.

16

Binding Property: getter, setter, and property getter

```java
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

```java
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

17

Unidirectional Binding

```java
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());
  }
}
```

BindingDemo

Run

18

Bidirectional Binding

- Occasionally, it is useful to synchronize two properties so that a change in one property is reflected in another object, and vice versa. This is called a *bidirectional binding.*
- If the target and source are both binding properties and observable properties, they can be bound bidirectionally using the **bindBidirectional** method.

BidirctionalBindingDemo

Run

19

14.6 Common Properties and Methods for Nodes

- JavaFX style properties are called *JavaFX CSS.* In JavaFX, a style property is defined with a prefix **–fx-.**

  http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html

- The abstract ***Node*** class defines many properties and methods that are common to all nodes.
- *style*: set a JavaFX CSS style
  - The syntax for setting a style is **styleName:value.** Multiple style properties for a node can be set together separated by semicolon (;).
- *rotate*: Rotate a node

NodeStyleRotateDemo

Run

20

14.7 The Color Class

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

javafx.scene.paint.Color

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

The Color Class

- JavaFX defines the abstract **Paint** class for painting a node. The **javafx.scene.paint.Color** is a concrete subclass of **Paint**, which is used to encapsulate colors. *The **Color** class can be used to create colors.*
- RGBA model: RGBA stands for red, green, blue, and alpha.
  - 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 alpha value indicates the opacity. It defines the transparency of a color within the range from 0.0 (completely transparent) to 1.0 (completely opaque).
- Many standard colors are defined as constants in the **Color** class. e.g. Color.RED

14.8 The Font Class

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

javafx.scene.text.Font

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

14.8 The Font Class

- *A **Font** describes font name, weight, and size.*
- You can set fonts for rendering the text. The **javafx.scene.text.Font** class is used to create fonts.
- A Font is defined by its name, weight, posture, and size.
  - The font postures are two constants: **FontPosture.ITALIC** and **FontPosture.REGULAR.**
- A Font instance can be constructed using its constructors or using its static methods.

FontDemo | Run

14.9 The Image and ImageView Classes

- The **javafx.scene.image.Image** class represents a graphical image and is used for loading an image from a specified filename or a URL.

  new Image("image/china.gif")

  new Image("http://www.cs.armstrong.edu/liang/image/china.gif")

- The **javafx.scene.image.ImageView** is a node for displaying an image. An ImageView can be created from an Image object, a file or a URL.

  Image image = **new Image("image/china.gif");**

  ImageView imageView = **new ImageView(image);**

  ImageView imageView = **new ImageView("image/china.gif");**

The Image Class

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

javafx.scene.image.Image

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

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

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

26

The ImageView Class

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.

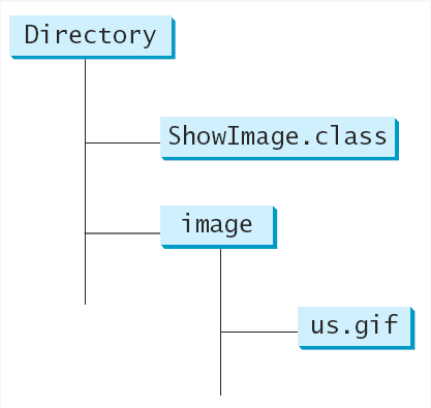
Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.

ShowImage | Run

27

Note

- Note that you must place the image file in the same directory as the class file.
  - Eclipse: project\bin\
- If you use the URL to locate the image file, the URL protocal **http://** must be present.




```
new Image("http://liveexample.pearsoncmg.com/book/image/us.gif");
```

14.10 Layout Panes and Groups

- Panes and groups are the containers for holding nodes.
- The **Group** class is often used to group nodes and to perform transformation and scale as a group. Panes and UI control objects are resizable, but group, shape, and text objects are not resizable.
- JavaFX provides many types of panes for organizing nodes in a container.

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

FlowPane

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

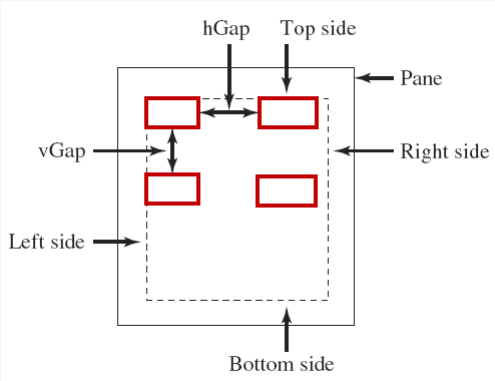
The horizontal gap between the nodes (default: `0`).
The vertical gap between the nodes (default: `0`).

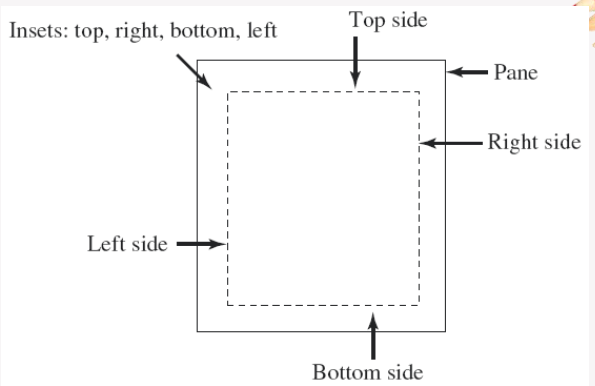
Creates a default `FlowPane`.
Creates a `FlowPane` with a specified horizontal and vertical gap.

Creates a `FlowPane` with a specified orientation.

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

FlowPane





ShowFlowPane

Run

31

GridPane

javafx.scene.layout.GridPane

```
-alignment: ObjectProperty<Pos>
-gridLinesVisible:
   BooleanProperty
-hgap: DoubleProperty
-vgap: DoubleProperty
```

```
+GridPane()
+add(child: Node, columnIndex:
   int, rowIndex: int): void
+addColumn(columnIndex: int,
   children: Node...): void
+addRow(rowIndex: int,
   children: Node...): void
+getColumnIndex(child: Node):
   int
+setColumnIndex(child: Node,
   columnIndex: int): void
+getRowIndex(child:Node): int
+setRowIndex(child: Node,
   rowIndex: int): void
+setHalighnment(child: Node,
   value: HPos): void
+setValighnment(child: Node,
   value: VPos): void
```

The 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 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.

ShowGridPane

Run

32

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`.

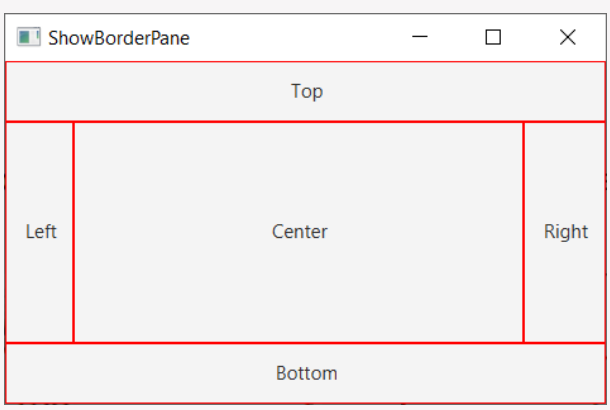
ShowBorderPane

Run

33

BorderPane

- The **BorderPane** places the nodes in five regions of the pane.



HBox

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.

35

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.

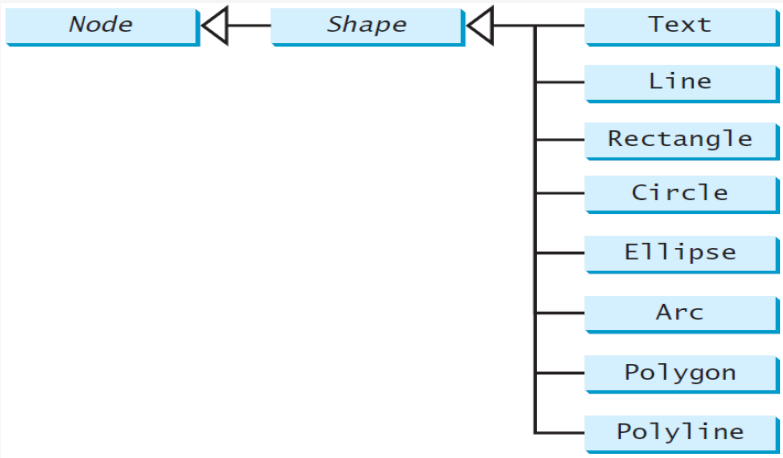
ShowHBoxVBox

Run

36

14.11 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<Font>
```

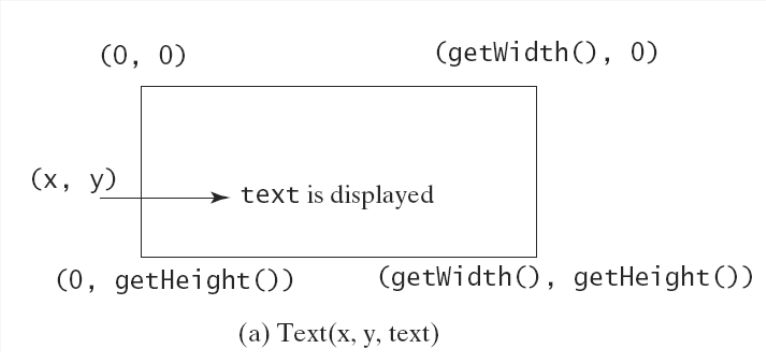
```
+Text()
+Text(text: String)
+Text(x: double, y: double,
   text: String)
```

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



(a) Text(x, y, text)

ShowText

Run

Line

javafx.scene.shape.Line

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

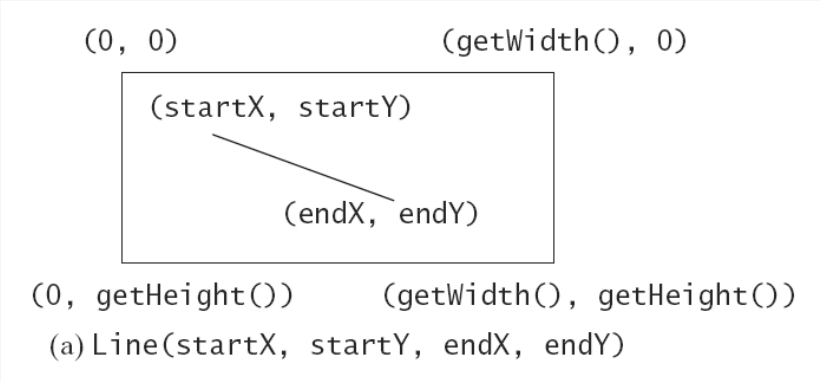
```
+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

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 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.

Line Example



(a) `Line(startX, startY, endX, endY)`

ShowLine

Run

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).

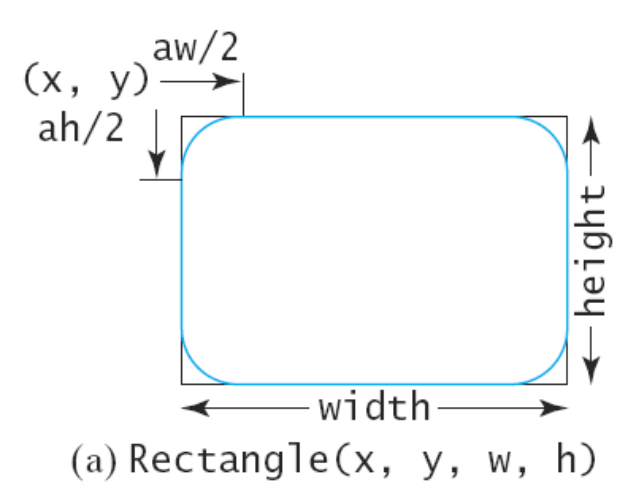
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

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



ShowRectangle

Run

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
```

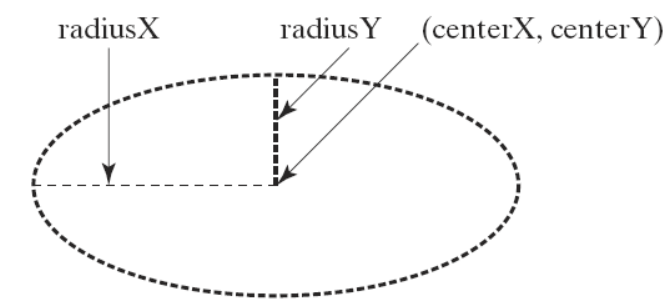
```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

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.

Ellipse



(a) `Ellipse(centerX, centerY, radiusX, radiusY)`

ShowEllipse

Run

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

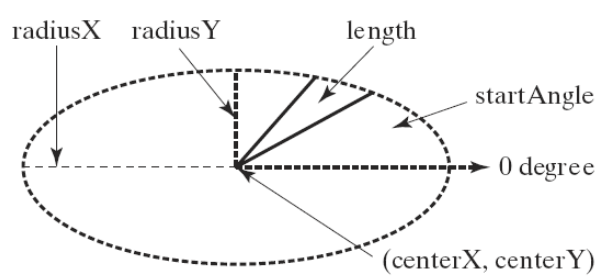
```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

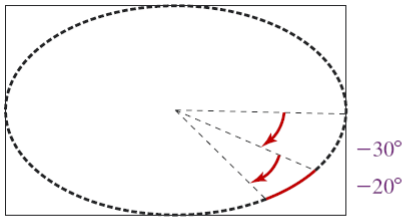
The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.

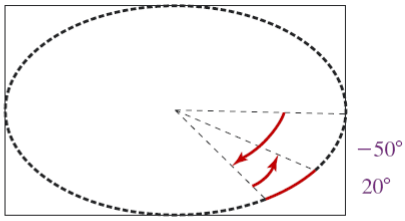
Arc Examples



(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`



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

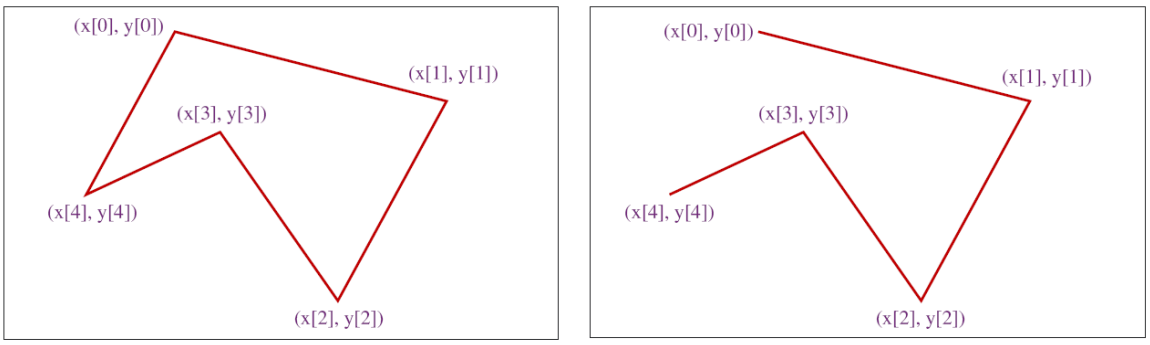


(b) Negative starting angle −50° and positive spanning angle 20°

ShowArc

Run

Polygon and Polyline



(a) Polygon (b) Polyline

Polygon

javafx.scene.shape.Polygon

```
+Polygon()
+Polygon(double... points)
+getPoints():
   ObservableList<Double>
```

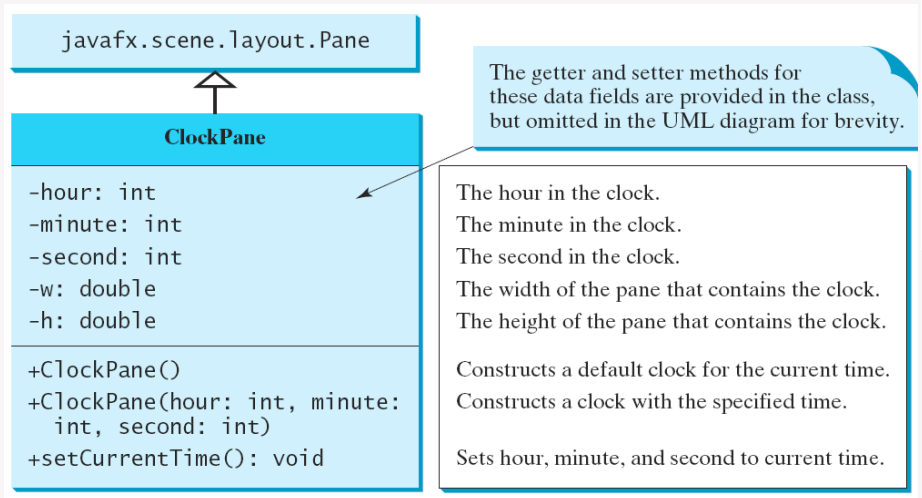
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

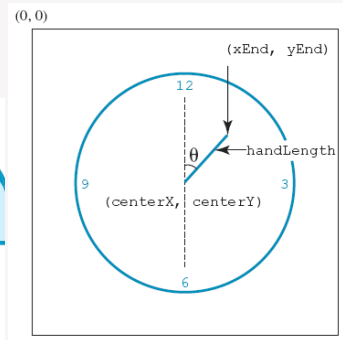
Creates an empty `Polygon`.
Creates a `Polygon` with the given points.
Returns a list of double values as x-and y-coordinates of the points.

ShowPolygon

Run

Case Study: The ClockPane Class





ClockPane

DisplayClock

Run

51

Chapter Summary

Chapter Summary

- JavaFX is the new framework for developing rich Internet applications. JavaFX completely replaces Swing and AWT.
- A main JavaFX class must extend ***javafx.application.Application*** and implement the ***start*** method. The primary stage is automatically created by the JVM and passed to the start method.
- A stage is a window for displaying a scene. You can add nodes to a scene. Panes, controls, and shapes are nodes. Panes can be used as the containers for nodes.
- The **Node** class defines many properties that are common to all nodes. You can apply these properties to panes, controls, and shapes.

Chapter Summary

- JavaFX provides many types of panes for automatically laying out nodes in a desired location and size. The ***Pane*** is the base class for all panes. It contains the ***getChildren()*** method to return an ***ObservableList***. You can use *ObservableList's* ***add(node)*** and ***addAll(node1, node2, ...)*** methods for adding nodes into a pane.
- A ***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. A ***GridPane*** arranges nodes in a grid (matrix) formation. The nodes are placed in the specified column and row indices. A ***BorderPane*** can place nodes in five regions: top, bottom, left, right, and center. An ***HBox*** lays out its children in a single horizontal row. A ***VBox*** lays out its children in a single vertical column.

Chapter Summary

- You can create a ***Color*** object with the specified red, green, blue components, and opacity value.
- You can create a ***Font*** object and set its name, size, weight, and posture.
- The ***javafx.scene.image.Image*** class can be used to load an image and this image can be displayed in an ***ImageView*** object.
- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.

Programming Exercises

1, 2, 3, 8, 9, 10, 11,

12, 18, 20, 26, 27