§14.1 Introduction

- JavaFX is a framework and (large) set of objects we can use to develop GUI-based applications
- The JavaFX API is a good example of how object-oriented principles can be applied in software development
- This chapter introduces us to *creating* the UI, but it will be very simple, and it won't actually *do* anything.
- Chapter 15 (Event-Driven Programming) shows us how to get that interface to *do* things (how to link executable code to the UI elements)
- Chapter 16 shows us more elements we can use to add different kinds of functionality to the UI

§14.2: JavaFX vs Swing vs AWT

- Java was first released with GUI support in something called the Abstract Windows Toolkit (AWT). AWT had some particular problems with how it was implemented on some platforms.
- AWT was replaced by Swing, which was more flexible. Swing was designed primarily for use in desktop applications. Swing has now been replaced by a new GUI library called JavaFX.
- JavaFX lets us write applications can run on a desktop and from a Web browser.
- 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.

§14.3: JavaFX Programs: Basic Structure

- JavaFX programs all start not as some “regular” class like we’ve been doing, but as an extension of the abstract Application class in JavaFX, javafx.application.Application

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

Becomes:

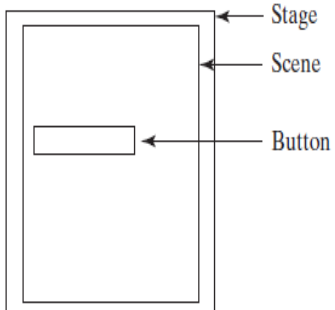
```
import javafx.application.Application;
…
public class MyProgram extends Application
{
    // Body of class
}
```

§14.3: JavaFX Programs: Basic Structure

- JavaFX programs are based on the analogy of a stage (think “theater stage” for the moment).
- On the stage are scenes, and each scene is also made up of other components.
- On a theater stage, the stage may be divided into portions, where individual scenes take place.
- Each scene’s set will have actors, props, backdrops, lighting, etc.
- In JavaFX, we create the components, add them to scenes, and then add scenes to the stage

§14.3: JavaFX Programs: Basic Structure

- 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
- Just as a music festival may have simultaneous performances on multiple stages, we can have more than one stage (window) in our programs.
- The code to set up this two-stage UI is on the next slide
- 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)`



§14.3: 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 scene and place a button in the scene
    Button btOK = new Button("OK");
    Scene scene = new Scene(btOK, 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

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

override start

create a button

create a scene

set stage title

set a scene

display stage

main method

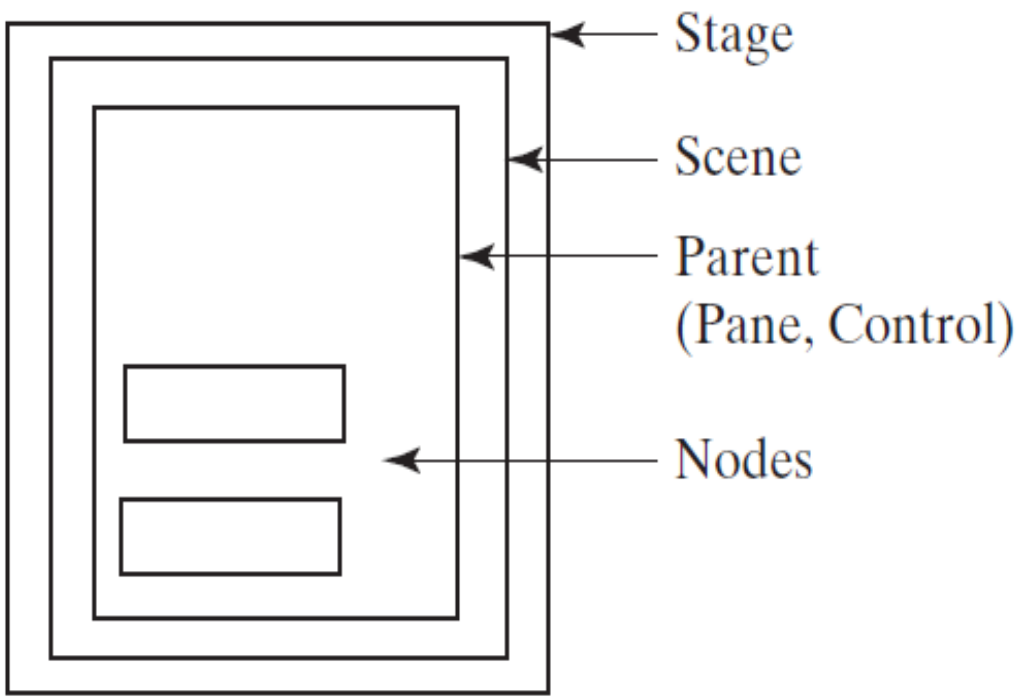
launch application

§14.3: A Program displays Multiple stages


```
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);      // Put the scene in the stage
        primaryStage.show();               // Display the primary stage

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

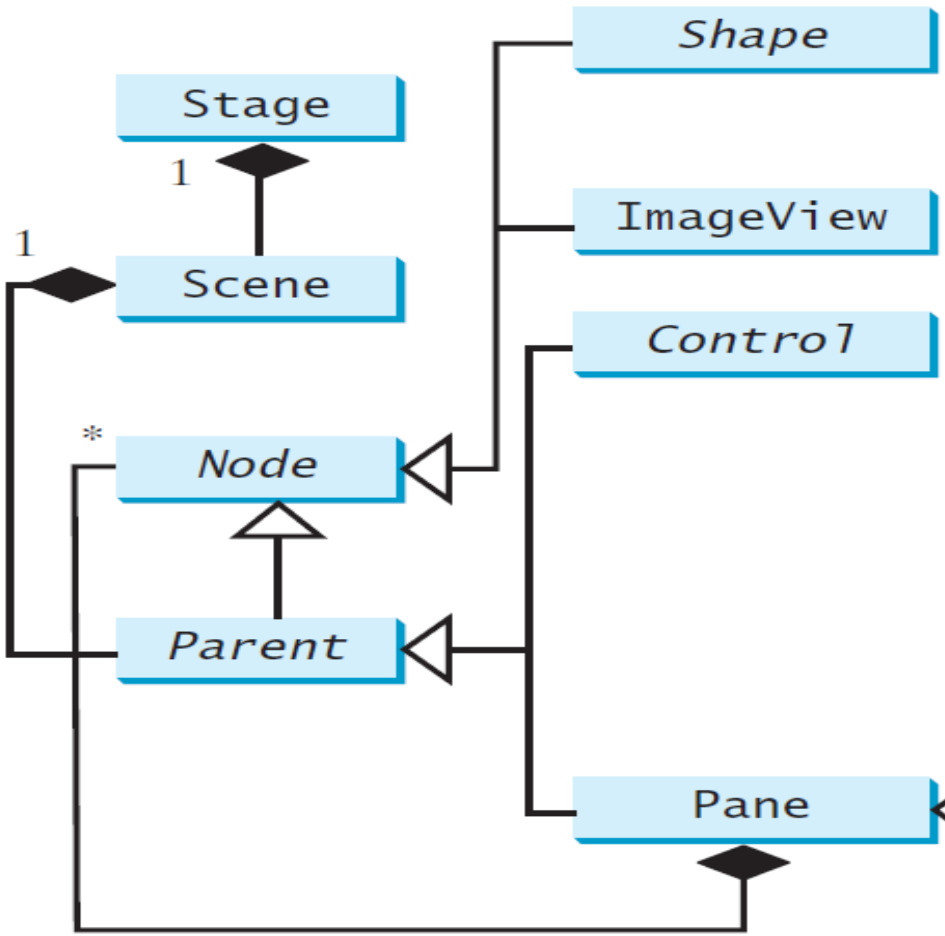
§14.4: Panes, UI Controls, and Shapes

- In Listing 14.1, 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.



§14.4: Panes, UI Controls, and Shapes

- Panes can even contain other panes:



Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

§14.4: Panes, UI Controls, and Shapes

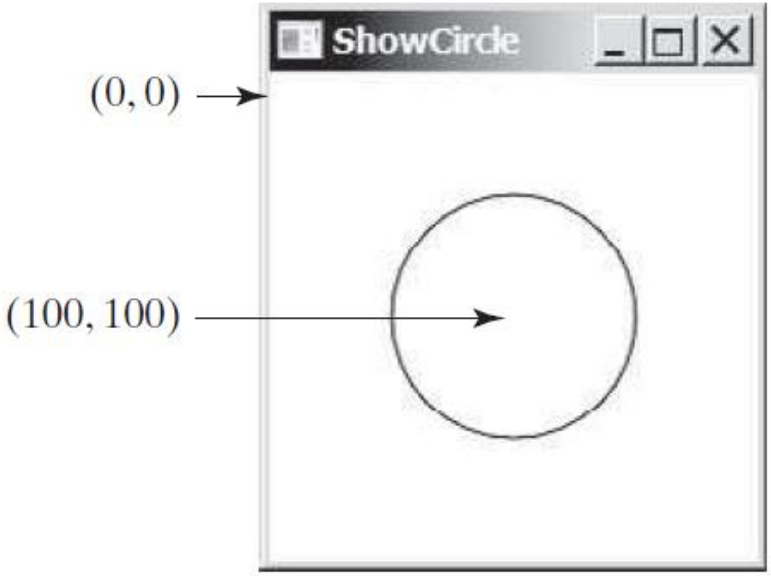
- The following slide shows the code to create this version of the same UI, with a single button inside a pane (so that the button doesn't occupy the whole stage).
- It uses a StackPane (which we'll discuss later).
- In order to add something to a pane, we need to access the list of things in the pane, 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

§14.4: Panes, UI Controls, and Shapes


```
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) {
     StackPane pane = new StackPane(); // Make a pane to work with
    // create a new button, and add it to the pane's list of children
        pane.getChildren().add(new Button("OK"));

        // Make a new scene, containing the pane
        Scene scene = new Scene(pane, 200, 50);
    primaryStage.setTitle("Button in a pane"); // Set the stage title
    primaryStage.setScene(scene);            // Put scene in the stage
    primaryStage.show();                    // Display the stage
    }
}
```

§14.4: Panes, UI Controls, and Shapes

- Before we can do much with shapes, we have to talk about coordinates within a 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. Visually, we're in Cartesian quadrant IV, but Y stays positive.
- All coordinates are in pixels



§14.4: Panes, UI Controls, and Shapes

```
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 200-200 scene from the pane, and place the scene 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
    } }
```

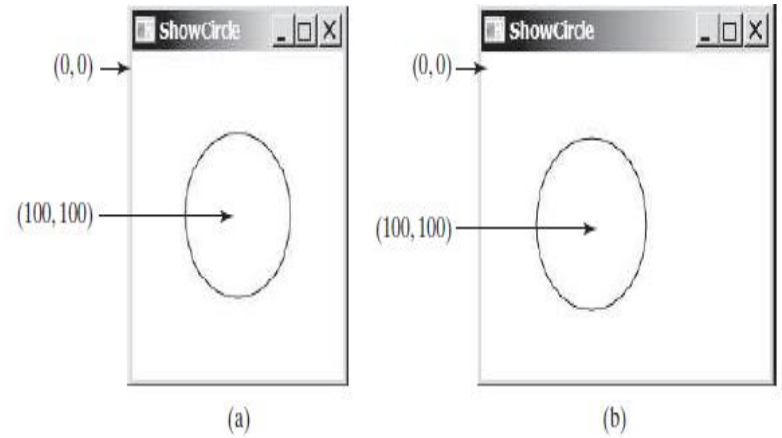


FIGURE 14.5 (a) A circle is displayed in the center of the scene. (b) The circle is not centered after the window is resized.

§14.5 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 x or y value of the circle's center.
- This is what *property binding* is all about
- The target object (called the binding object) gets bound to the source object (the bindable object). When there's a change to the source, it gets automatically sent to the target.
- The binding syntax is `target.bind(source);`
- The following listing (14.5) shows how to bind the circle's X and Y center values to the pane's width (for clarity, the `import` statements at the top have been omitted)

§14.5 Property Binding

```
public class ShowCircleCentered extends Application {
  @Override // Override the start method
    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 stage title
  primaryStage.setScene(scene);            // Put scene in stage
  primaryStage.show();                    // Display the stage
    }
}
```

§14.7 The `Color` Class

- Images are made up of pixels, and each pixel is made up of a red, a green, and a blue component, and that mixing these three components allows us to produce a huge number of colors
- JavaFX uses this same color model, but adds a fourth "component" to a pixel – its *opacity*. غموض
- There are 3 sets of color constructors ("mixers"):
  - The ones named Color (or color) require double values ∈ [0.0, 1.0] for the R/G/B/A components
  - The ones named rgb require int values ∈ [0, 255]
- Just like String, Color is immutable.
- If we want a lighter version of this Color, we can use .lighter(), but we get a NEW color, rather than changing the value of the current color, just like .toUpperCase doesn't *change* a String; it gives us a new one with upper case characters.

§14.7 The `Color` Class

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

- The `color()` and `rgb()` methods are static, and don't require instantiating a color
- Also, we can use named color constants: `BEIGE`, `BLACK`, `BLUE`, `BROWN`, `CYAN`, `DARKGRAY`, … There are about 150 of them.

§14.8: The `Font` Class

- A **Font** instance can be constructed using its constructors or using its static methods. A full **Font** is defined by its name, weight, posture, and size.
- Font font1 = **new** Font("**SansSerif**", **16**);
- Font font2 = Font.font("**Times New Roman**", FontWeight.BOLD, FontPosture.ITALIC, **12**);

`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()`

14.8 A program displays a Label using a Font

create a StackPane

create a Circle

create a Color

add circle to the pane

create a label

create a font

add label to the pane

```
10  public class FontDemo extends Application {
11    @Override // Override the start method in the Application class
12    public void start(Stage primaryStage) {
13      // Create a pane to hold the circle
14      Pane pane = new StackPane();
15
16      // Create a circle and set its properties
17      Circle circle = new Circle();
18      circle.setRadius(50);
19      circle.setStroke(Color.BLACK);
20      circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
21      pane.getChildren().add(circle); // Add circle to the pane
22
23      // Create a label and set its properties
24      Label label = new Label("JavaFX");
25      label.setFont(Font.font("Times New Roman",
26        FontWeight.BOLD, FontPosture.ITALIC, 20));
27      pane.getChildren().add(label);
28
29      // Create a scene and place it in the stage
30      Scene scene = new Scene(pane);
31      primaryStage.setTitle("FontDemo"); // Set the stage title
32      primaryStage.setScene(scene); // Place the scene in the stage
33      primaryStage.show(); // Display the stage
34    }
35  }
```

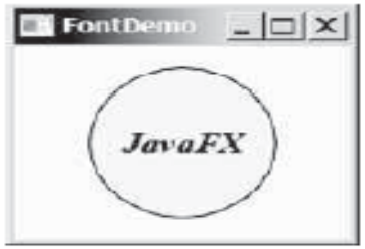
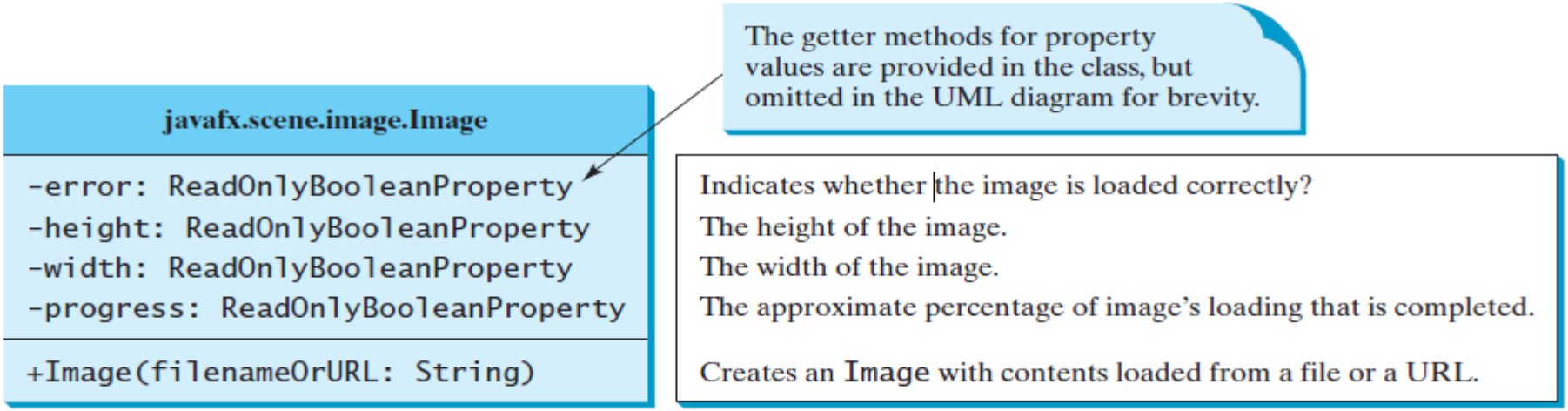


FIGURE 14.11 A label is on top of a circle displayed in the center of the scene.

§14.9: 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
  - we use the ImageView node (and attach it to a scene)
- 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



§14.9: The Image & ImageView Classes

- The ImageView class:

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

javafx.scene.image.ImageView

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

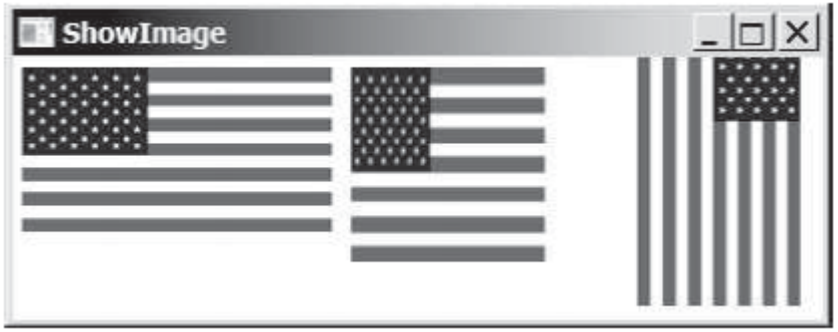
```
+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
```

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

an ImageView to display an image

§14.9: The Image & ImageView Classes

```
// Bunch of omitted Import statements, Class header, and start() header
{
    // This is just the BODY of start()
    // Create a pane to hold the image views
    Pane pane = new HBox(10);                // HBox is covered in next section
    pane.setPadding(new Insets(5, 5, 5, 5));

    Image image = new Image("image/us.gif");        // Load image from file

    pane.getChildren().add(new ImageView(image)); // 1st IV gets image as-is

    ImageView imageView2 = new ImageView(image);  // 2nd IV forces image to
    imageView2.setFitHeight(100);                 // fit into 100-x-100
    imageView2.setFitWidth(100);                  // pixel area
    pane.getChildren().add(imageView2);

    ImageView imageView3 = new ImageView(image);  // 3rd IV leaves size as-is,
    imageView3.setRotate(90);                     // but rotates (CW) 90 deg,
    pane.getChildren().add(imageView3);

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



§14.9: 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)

- Notes:
  - `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

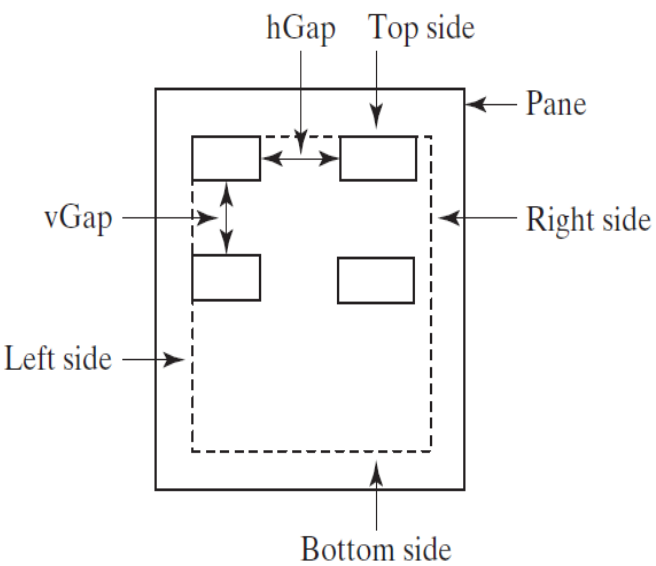
§14.10: Layout Panes

- As we've said, we add our `Nodes` to a `Pane`, and then add the `Pane` to a `Scene`, and then the `Scene` to a `Stage`.
- How do we arrange (i.e., lay out) the `Nodes` on the pane?
- Java has several different kinds of `Panes` that do a lot of the layout work for us. We've already used the `Pane` , `HBox`, and `StackPane`; we'll start with the `FlowPane`…

Name	Description
`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
`StackPane`	Places nodes on top of each other in the center of the pane
`FlowPane`	Places the notes row-by-row horizontally or column-by-column vertically (reading order)
`GridPane`	Provide a 2-D grid of cells, into which we can place nodes
`BorderPane`	Divides pane into top, bottom, left, right, and center regions
`HBox`	Places nodes in a single (horizontal) row
`VBox`	Places nodes in a single (vertical) column

§14.10.1: The FlowPane

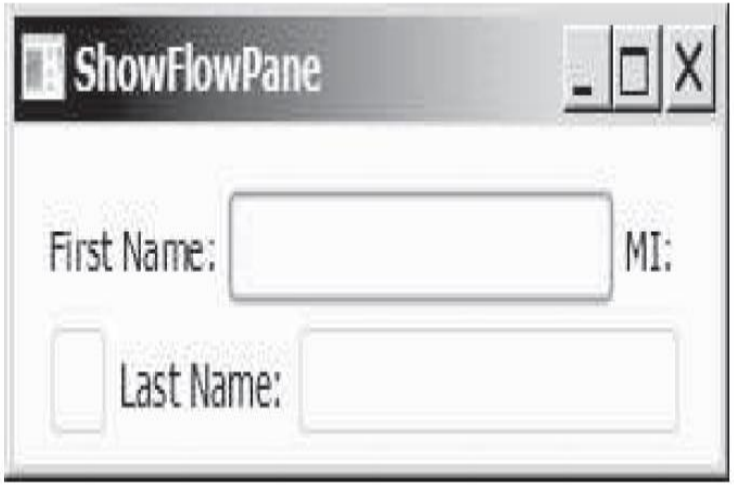
- The FlowPane can be set up to work in "reading order" (sequential rows left-to-right), by using Orientation.HORIZONTAL or in sequential top-to-bottom columns (Orientation.VERTICAL)
- You can also specify the gap between nodes (in pixels)
- So far, we have seen putting Button and ImageView nodes on a pane.



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

§14.10.1: 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);

   // 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 st
   Scene scene = new Scene(pane, 200, 250);
   primaryStage.setTitle("ShowFlowPane"); /
   primaryStage.setScene(scene);          /
   primaryStage.show();                   /
}
```



§14.10.1: 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());
```

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").

}

§14.10.1: 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());
```

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

§14.10.1: 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());
```

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

```
}
```

§14.10.2: The `GridPane`

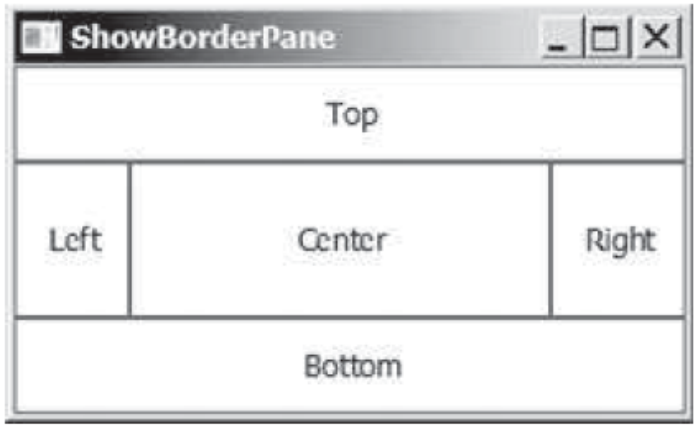
- The `GridPane` divides the pane's area into a 2-D grid of rows and columns.
- Listing 14.11 shows the previous example redone, with the three `Labels` in the left column, and the three `TextFields`
- A few notes:
  - The "Add" button is right-aligned within its cell (l. 31)
  - The whole frame is centered (l. 17)
  - The labels get the default horizontal alignment of "left"
  - We specify the column first (backwards from arrays)
  - Not every cell needs to be filled
  - Elements can be moved from one cell to another

§ Listing 14.11 : The GridPane

create a grid pane
set properties
add label
add text field
add button
align button right
create a scene
display stage

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

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

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

§14.10.3: The BorderPane



- The BorderPane divides the pane into five “regions”
- The program in Listing 14.12 places a CustomPane in each region.
- The CustomPane is an extension of StackPane, which is used to display the labels
- 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)

§ Listing 14.12: The BorderPane

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

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

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

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

create a border pane

add to top

add to right

add to bottom

add to left

add to center

define a custom pane

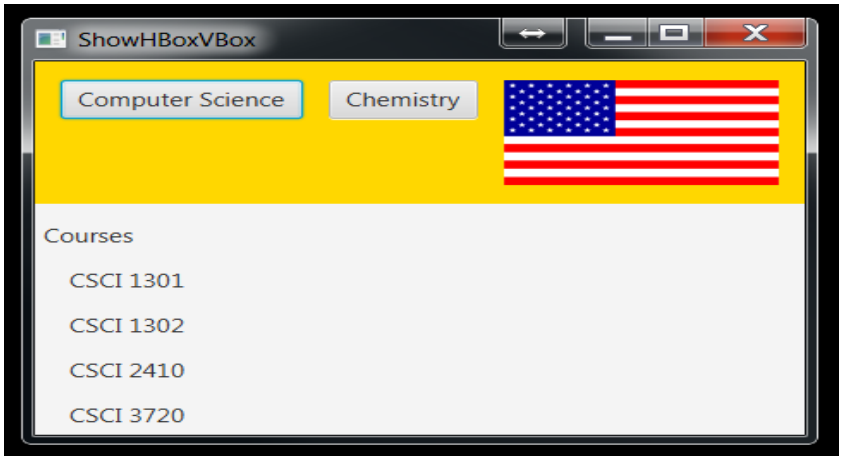
add a label to pane

set style

set padding

§14.10.4: HBox and VBox

- The `FlowPane` gave us rows and columns (in reading order)
- The `HBox` and `VBox` panes give us a single row or column (respectively)
- The program in Listing 14.13 (p. 559) illustrates the use of a `BorderPane`, an `HBox`, and a `VBox`
- It creates a `BorderPane` with only the `Top` and `Left` regions used (the others are empty)
- The `Top` region contains an `HBox` with two buttons and an `ImageView`
- The `Left` region contains a `VBox` with 5 labels



§ Listing 14.13 : HBox and VBox

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

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

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

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

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

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

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

    return vBox;
  }
}
```

create a border pane

add an HBox to top

add a VBox to left

create a scene

display stage

getHBox

add buttons to HBox

return an HBox

getVBox

add a label

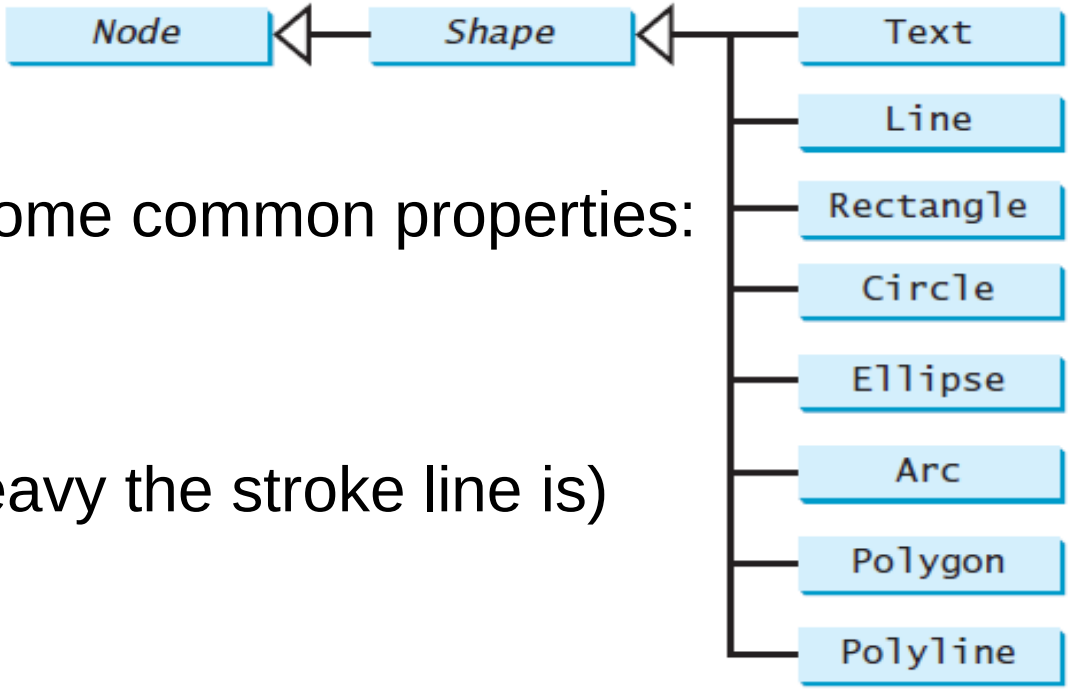
set margin

add a label

return vBox

§14.11: Shapes

- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.
- The **Shape** class is the abstract base class that defines the common properties for all shapes.



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

§14.11.1: Text

- The Text shape lets us put text on a pane
- They're two different kinds of text, with two different inheritance paths.
- Listing 14.14 (pp. 561 – 562) shows placing three `Text` shapes on a pane, each time using the (`double`, `double`, `String`) form of the constructor to specify the x and y locations for the String.
- The text we place as a Shape can have its color, font (typeface, weight, size, and posture), underline, and strikethrough properties set.
  - This is different from a Label's JavaFX CSS properties
- Once placed, we can move text by using `setX()` and `setY()` to give it a new location

§ Listing 14.14 : Text

create a pane

create a text
set text font

add text to pane

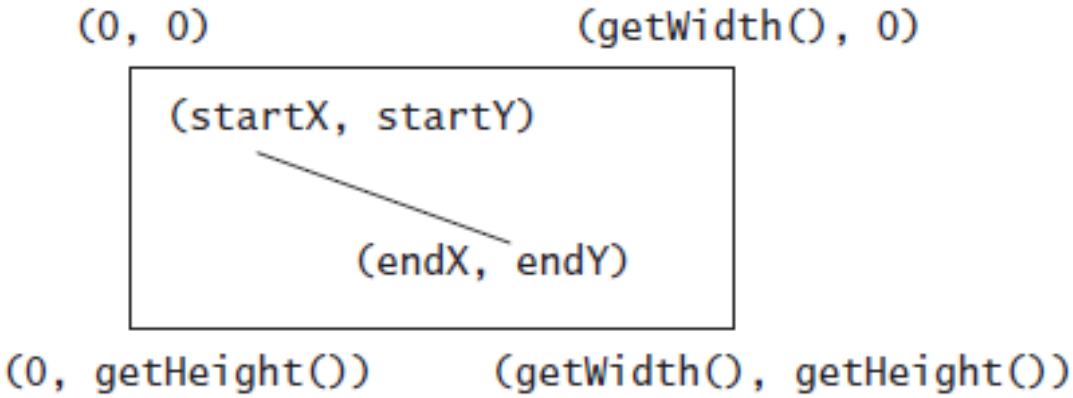
create a two-line text
add text to pane

create a text
set text color
set underline
set strike line
add text to pane

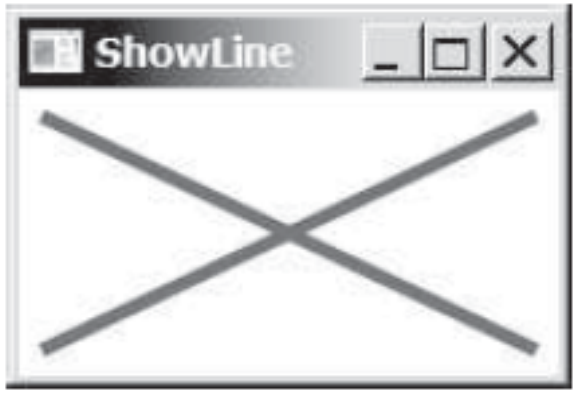

```
public class ShowText extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the texts
    Pane pane = new Pane();
    pane.setPadding(new Insets(5, 5, 5, 5));
    Text text1 = new Text(20, 20, "Programming is fun");
    text1.setFont(Font.font("Courier", FontWeight.BOLD,
      FontPosture.ITALIC, 15));
    pane.getChildren().add(text1);

    Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
    pane.getChildren().add(text2);

    Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
    text3.setFill(Color.RED);
    text3.setUnderline(true);
    text3.setStrikethrough(true);
    pane.getChildren().add(text3);

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

§14.11.2: Line

- Listing 14.15 (pp. 562 – 563) shows how to draw a pair of `Line` shapes on a pane
- Each line is specified by its two endpoints: `startX`, `startY` and `endX`, `endY` (all `double`s, in pixels)
- <u>Example:</u> Line line1 = **new** Line(**10**, **10**, **10**, **10**);



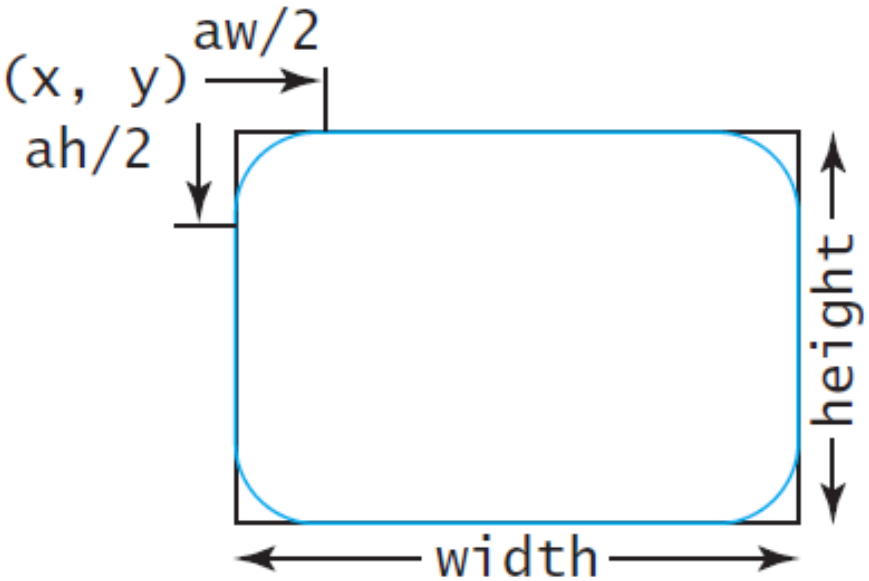
(a) Line(startX, startY, endX, endY)



(b) Two lines are displayed across the pane.

§14.11.3: Rectangle

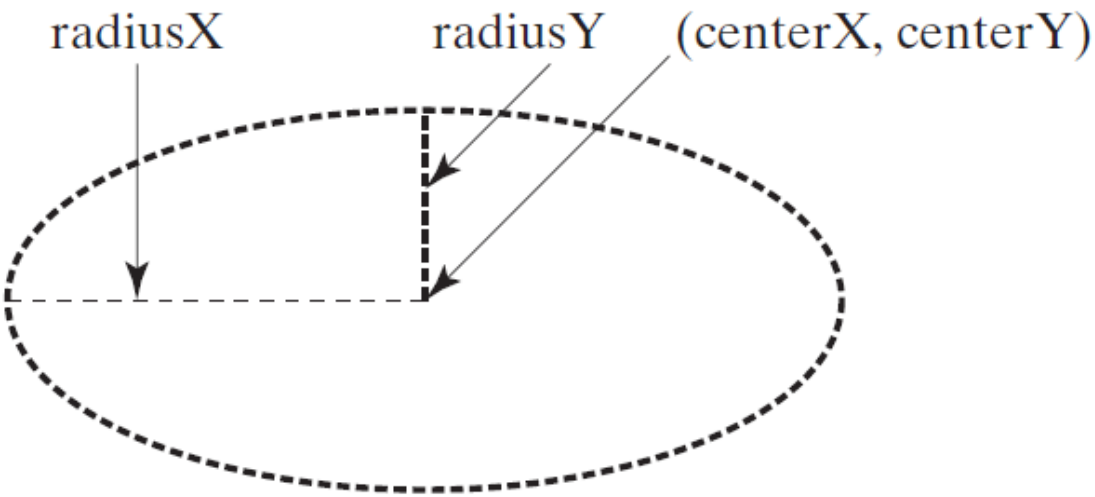
- The `Rectangle` shape is specified by its top-left corner (`x`, `y`) and its `width` and `height`.
- Optionally, we can specify the `arcWidth` and `arcHeight` (in pixels) of the corners, to create rounded corners (arc measurements of 0 make squared-off corners)
- <u>Example:</u> Rectangle r1 = **new** Rectangle(**25**, **10**, **60**, **30**);
- See Listing 14-16 and Figure 14.31



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

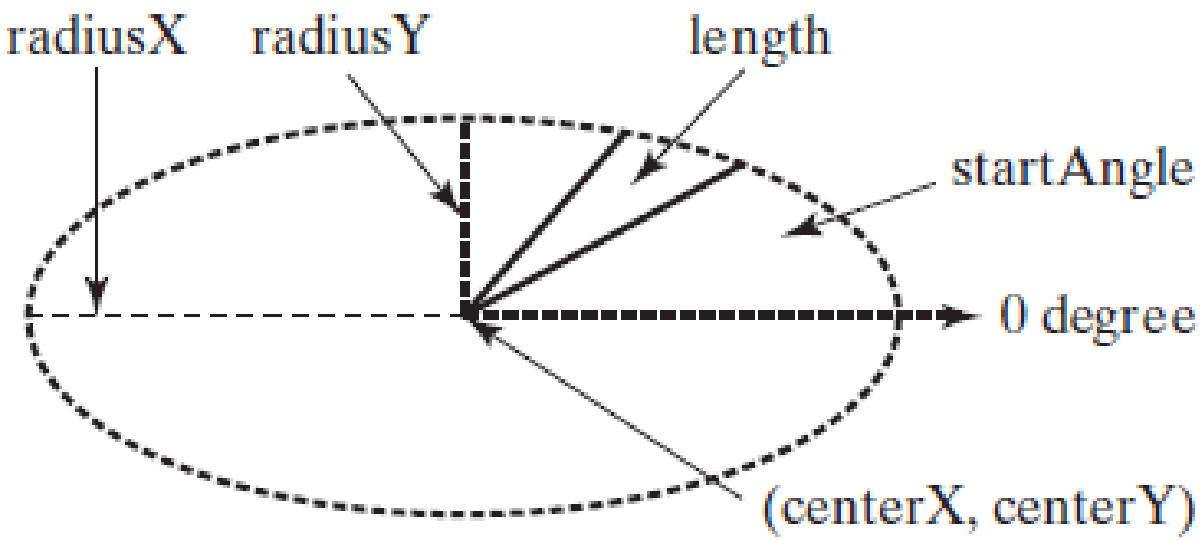
§14.11.4: `Circle` and `Ellipse`

- The `Circle` and `Ellipse` shapes are very similar, as you might expect
- Both are specified by their center point (`centerX`, `centerY`)
- The `Circle` has a single `radius`
- The `Ellipse` has *two* radii (`radiusX` & `radiusY`)
- See Listing 14.17 (pp. 566 – 7)
- <u>Example:</u> Ellipse e1 = **new** Ellipse(**150**, **100**, **100**, **50**);



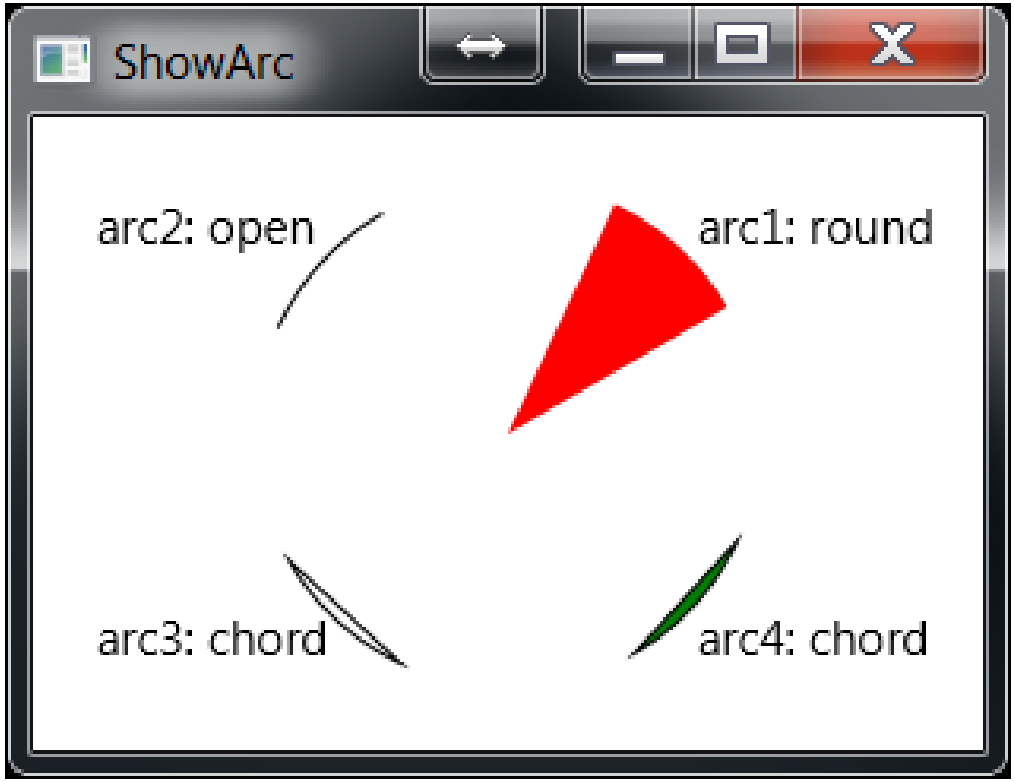
§14.11.5: Arc

- An `Arc` is just part of an `Ellipse`
- An `Arc` is specified by its center point, radii, start angle (in degrees), and length (in degrees)
  - If the two radii are equal, then it's a circular arc
  - Angles may be negative (clockwise sweep)
- <u>Example:</u> Arc arc1 = **new** Arc(**150**, **100**, **80**, **80**, **30**, **35**);



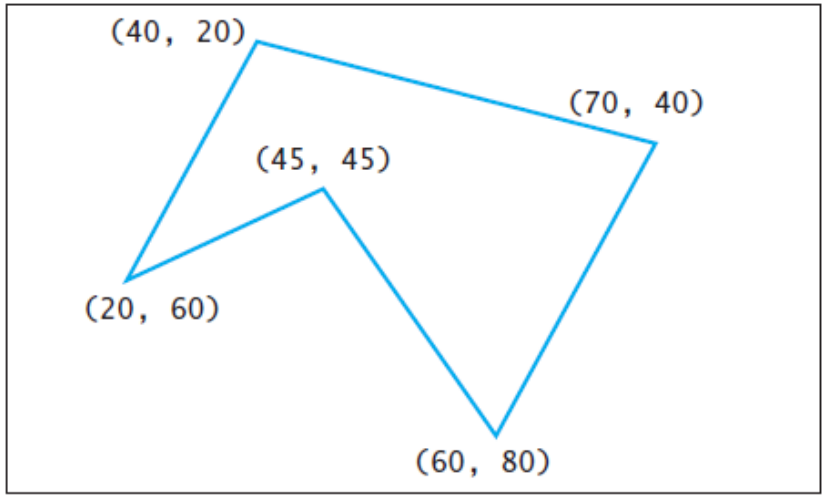
§14.11.5: Arc

- Once specified, an `Arc` can be drawn in any of three styles: `ArcType.OPEN`, `ArcType.CHORD`, or `ArcType.ROUND`
- See Listing 14.18 (pp. 568 – 569) for the code

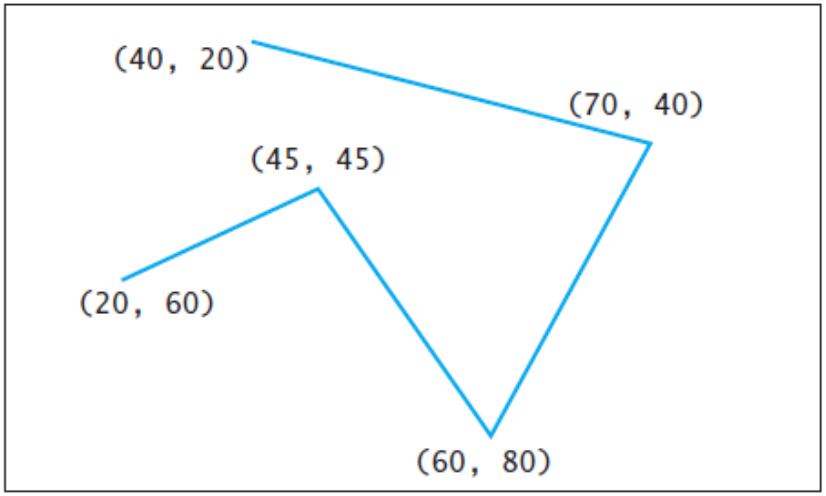


§14.11.6: Polygon and Polyline

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



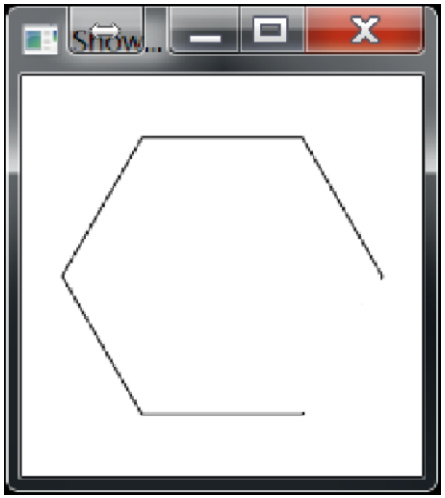
(a) Polygon

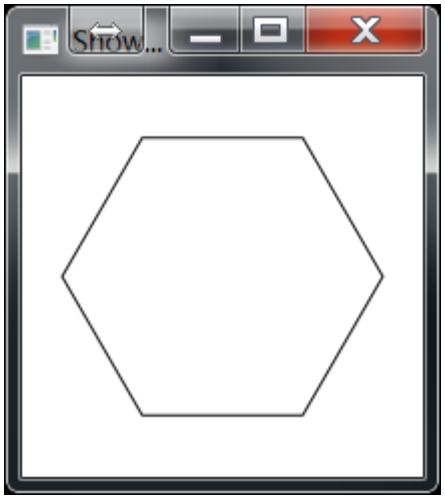


(b) Polyline

§14.11.6: Polygon and Polyline

- If we know all of the points up-front, we can specify them as a variable-length parameter list:
  `Polygon p = new Polygon(x1, y1, x2, y2, …);`
- If we don't know all of the points *a priori*, the `Polygon` exposes an `ObservableList` to which we can add points (i.e., generate the points on-the-fly and add them to the list as we go)
  - This latter approach is taken by the program to create a hexagon (Listing 14.19, pp. 570 – 1, and next slide)





§14.11.6: Polygon and Polyline

```
Pane pane = new Pane();                // create a pane
Polygon polygon = new Polygon();       // create a polygon
pane.getChildren().add(polygon);       // add the polygon to the pane
polygon.setFill(Color.WHITE);          // polygon will be filled in white
polygon.setStroke(Color.BLACK);        //     with a black border
ObservableList<Double> list = polygon.getPoints(); // get its vertex list

final double WIDTH = 200, HEIGHT = 200;            // See Fig. 14.40 (a)
double centerX = WIDTH / 2, centerY = HEIGHT / 2;  //
double radius = Math.min(WIDTH, HEIGHT) * 0.4;     //

// Generated and add (x, y) points to the polygon's vertex list
for (int i = 0; i < 6; i++)
{
   list.add(centerX + radius * Math.cos(2 * i * Math.PI / 6));
   list.add(centerY - radius * Math.sin(2 * i * Math.PI / 6));
}

// Create a scene and place it in the stage
Scene scene = new Scene(pane, WIDTH, HEIGHT);
primaryStage.setTitle("ShowPolygon"); // Set the stage title
primaryStage.setScene(scene);         // Put scene in stage
primaryStage.show();                  // Display the stage
```