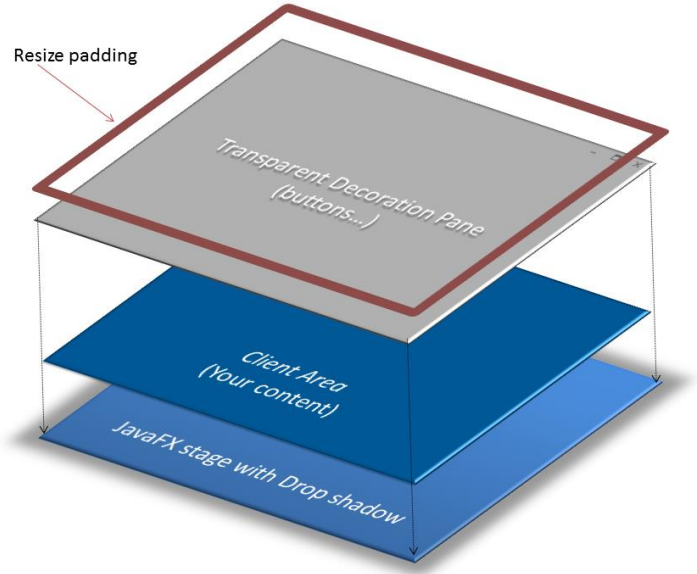
Graphical User Interfaces Programming

Shing-Chi Cheung
Computer Science and Engineering
HKUST



History: AWT → Swing → JavaFX

- AWT
  - When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT)
  - Limitation: AWT is prone to platform-specific bugs
- Swing
  - The AWT user-interface components were replaced since Java 2 by a more robust library known as Swing components
    - Swing components are painted directly on canvases using Java code.
    - They depend less on the target platform and use less native GUI resources
    - They are supported by Java Applets for graphics display on web browsers

History: AWT → Swing → JavaFX

- Swing
  - Limitations
    - Discontinued support of Java Applets on web browsers
    - Not designed to provide multi-touch and built-in 3D support

- JavaFX
  - Use the metaphor of a theater to model graphic applications
    - stage → scene → pane → node
  - Well integrated with the latest internet technologies, e.g., CSS
  - Take advantage of new hardware acceleration pipeline and graphics cards

https://www.youtube.com/watch?v=uxmhqv0in34

JavaFX

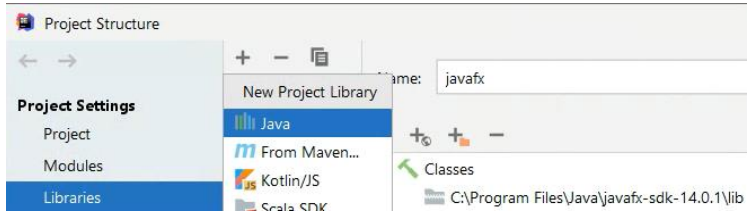
- JavaFX is the latest framework completely replacing AWT and Swing for GUI programs since Java 8
  - There will be no further development of AWT and Swing
- We will discuss the basics of JavaFX programming
- We will use JavaFX to demonstrate OOP. Specifically, we will introduce the framework of JavaFX and discuss JavaFX Graphical User Interface (GUI) components and their relationships.

Standalone JavaFX Module

- Java SDK is getting large in size over time
- Consensus to keep it down to a reasonable size
- Oracle has removed JavaFX from basic Java modules since Java 11 SDK
- Like Qt in C++, JavaFX is available as standalone Java modules, which can be supported by a third party (e.g., Gluon)
- Therefore, we need to explicitly include JavaFX modules in JavaFX applications

Configuring IntelliJ 2020.x to Run JavaFX

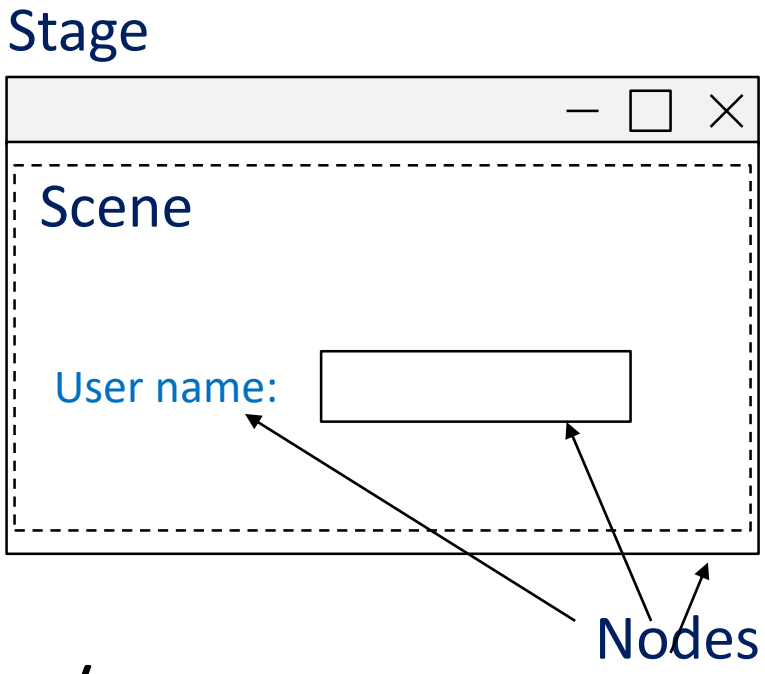
- Check if you are using Java 14 at *Project Structure->Project Settings*
- Check if JavaFX plugin is enabled at *Settings->Plugins*
- Download and install JavaFX 14 SDK to a folder at your PC
- Add the JavaFX lib to *Project Structure->Project Settings->Libraries*



- Define a new Path Variable *PATH_TO_FX* to point to JavaFX lib folder (e.g., c:\Program Files\Java\javafx-sdk-14.0.1\lib) at *Settings->Appearance & Behavior->Path Variables*
- Define *VM option* in *Run configuration*
  - --module-path ${PATH_TO_FX} --add-modules javafx.base,javafx.controls,javafx.fxml,javafx.media
- Install SceneBuilder at your PC
- Set the location of *SceneBuilder.exe* at *Settings->Languages & Frameworks->JavaFX*

Note: Please make sure you download the right version of javafx for the architecture of your machine.

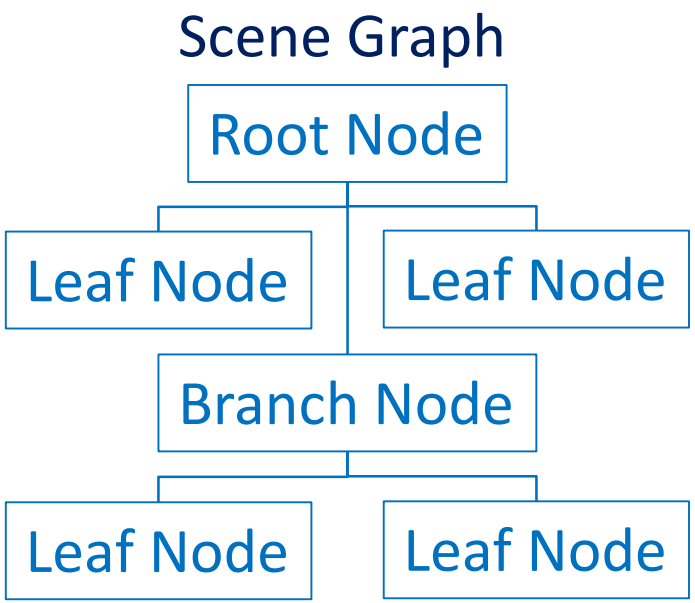
Basic Structure of JavaFX



- **Stage** (defined by javafx.stage.Stage)
  - The top-level container (i.e., window)
- **Scene** (defined by javafx.scene.Scene)
  - A container that carries individual controls / components
- **Node** (defined by javafx.scene.Node)
  - Specify a scene's content. Nodes are organized in an hierarchical Scene Graph

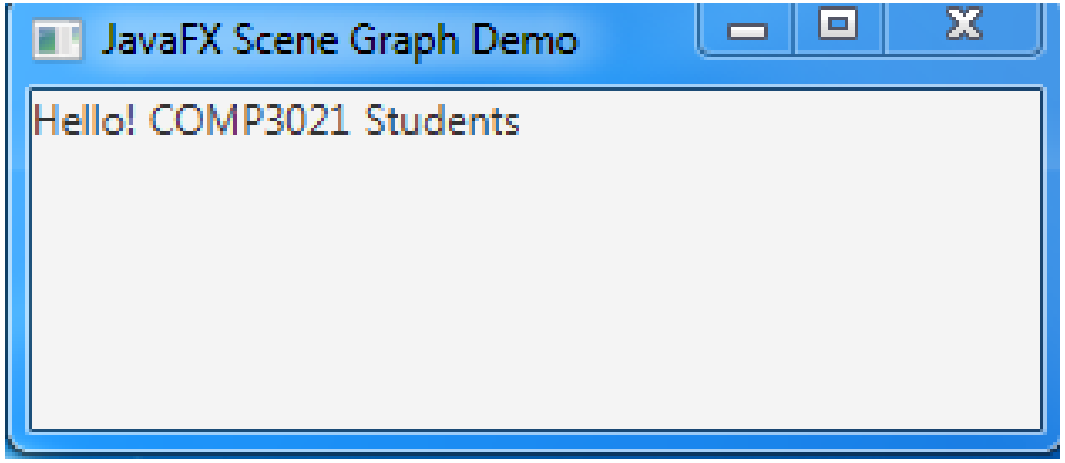
Scene Graph

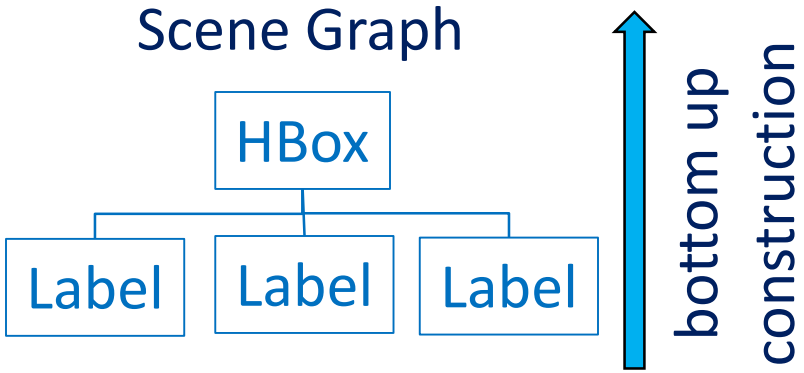
- A Scene Graph describes an hierarchy of graphical nodes to be display in a scene
- Nodes are visual / graphical objects:
  - Geometrical objects: 2D and 3D, e.g., circles, rectangles, etc.
  - UI controls: buttons, check boxes, choice boxes, text area, etc.
  - Layout panes: border pane, grid pane, flow pane, etc.
  - Media elements: audio, video and image objects



```java
public class JavaFXSceneGraphDemo extends Application {
  @Override
  public void start(Stage stage) {
    var label1 = new Label("Hello! ");
    var label2 = new Label("COMP3021 ");
    var label3 = new Label("Students");
    var root = new HBox();
    root.getChildren().add(label1);
    root.getChildren().add(label2);
    root.getChildren().add(label3);
    var scene = new Scene(root, 300, 100, Color.BLACK);
    stage.setTitle("JavaFX Scene Graph Demo");
    stage.setScene(scene);
    stage.show();
  }
  // create a stage object and pass it as an argument to the start method
  public static void main(String[] args) { launch(args); }
}
```

need to extend javafx.application.Application





JavaFXSceneGraphDemo.java

Four Steps to Construct an GUI

1. Define a class that extends abstract class Application
2. Override the abstract start method inherited from Application
3. In the overriding start method
   a. Build a scene graph
   b. Construct a scene with the root node of the scene graph
   c. Set up the stage with the constructed scene
4. Define a main method and launch the application

```
public class JavaFXSceneGraphDemo
extends Application {                    (1)
  @Override                              (2)
  public void start(Stage stage) {
    // build a scene graph
    // construct a scene with root node    (3)
    stage.setScene(scene);
    stage.show();
  }
  public static void main(String[] args) {
    launch(args);                         (4)
  }
}
```

primary stage

JavaFX Application Lifecycle

main → Launcher (or main) thread:

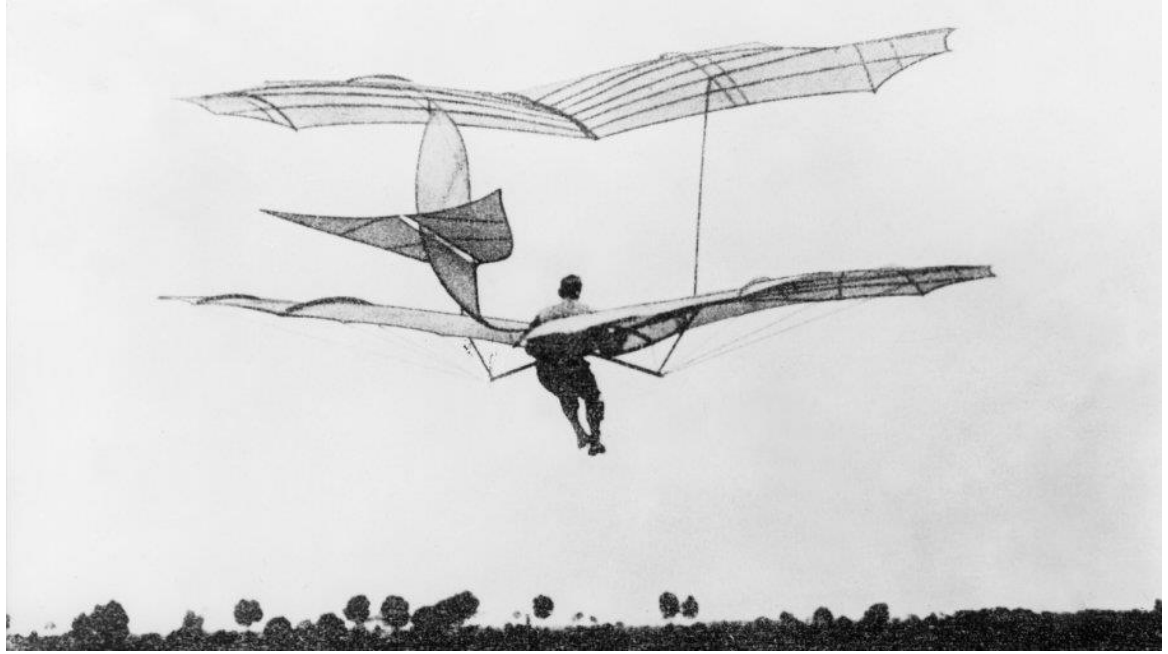
- Executes Application.launch(args)
  - Creates an instance o of the public subclass of Application
  - Executes o.init() to acquire resources
  - Creates an Application thread and passes to it the reference of o
  - Returns after Application thread has terminated

→ Application thread:

- Executes o.start(Stage)
- Listens to and handles events until:
  - It executes Platform.exit()
  - The last window has been closed
- Executes o.stop() to release resources acquired by o.init()
- Terminates

•Note that the start method in javafx.application.Application is abstract and must be overridden. The init and stop methods have concrete implementations that do nothing.
•Calling Platform.exit() is the preferred way to explicitly terminate a JavaFX Application. Directly calling System.exit(int) doesn't allow the stop() method to run.

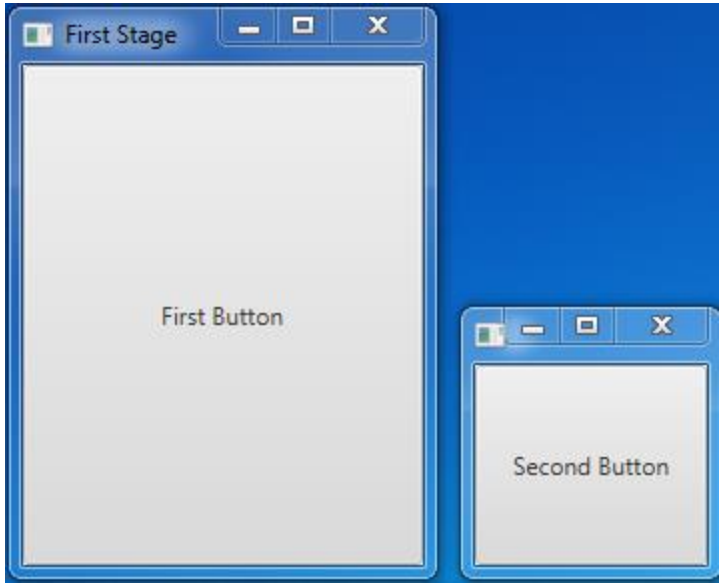
Our First Attempt



Building a Simple Application with Two Stages

```
// Step 1: extend Application
public class MultipleStageDemo extends Application {
  @Override
  // Step 2: override start
  public void start(Stage primaryStage) {
    // Step 3a: construct the root of a scene graph
    var button1 = new Button("First Button");
    // Step 3b: construct a scene
    var scene = new Scene(button1, 200, 250);
    // Step 3c: set up a stage
    primaryStage.setTitle("First Stage");
    primaryStage.setScene(scene);
    primaryStage.show();

      … // do similarly for the second stage
}
```

primary stage



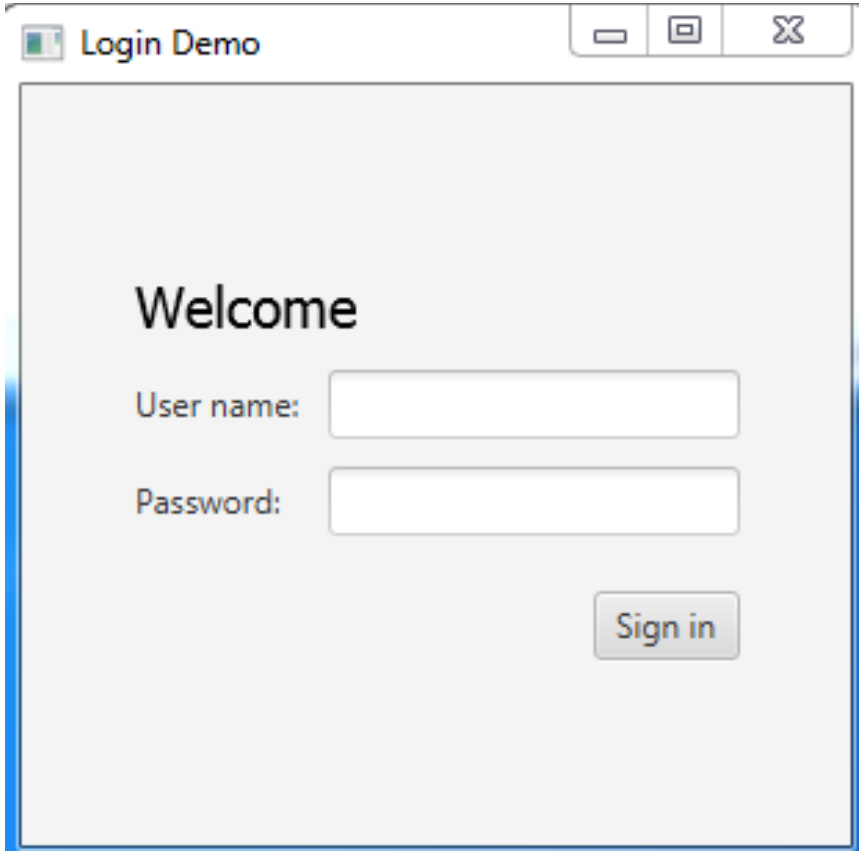

```
// Step 4: Define main method that launches
// the application
public static void main(String[] args) {
  launch(args);
}
}
```

MultipleStageDemo.java

Let’s Try Another GUI

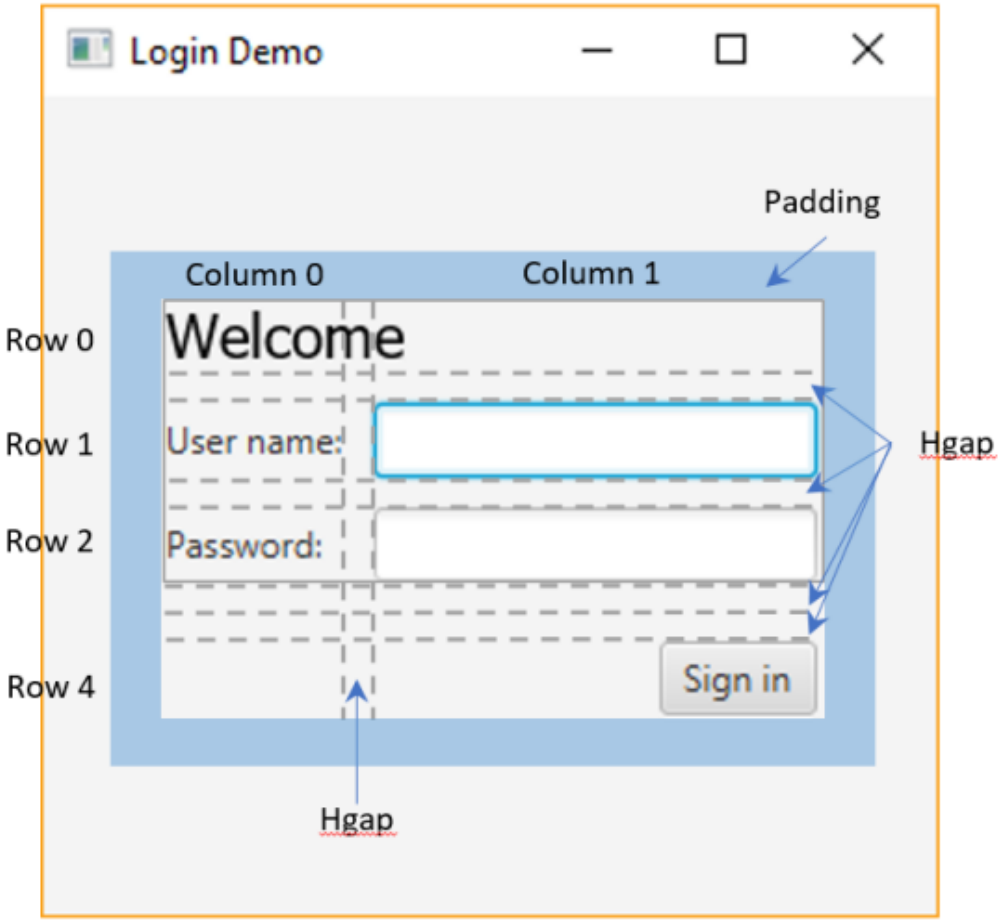
Building Another Application

- Let us try again by building a Login Demo application



Design

- Layout container: GridPane and Hbox
- UI controls: Label, Text, TextField, PasswordField, Button
- Helper classes: Font, Insets, Pos, FontWeight



Step 1: Import Statements & Extends Application

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.control.Label;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.control.TextField;
import javafx.scene.control.PasswordField;
import javafx.scene.control.Button;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.text.FontWeight;
```

This time we try out more JavaFX classes in additional to Application, Stage and Scene

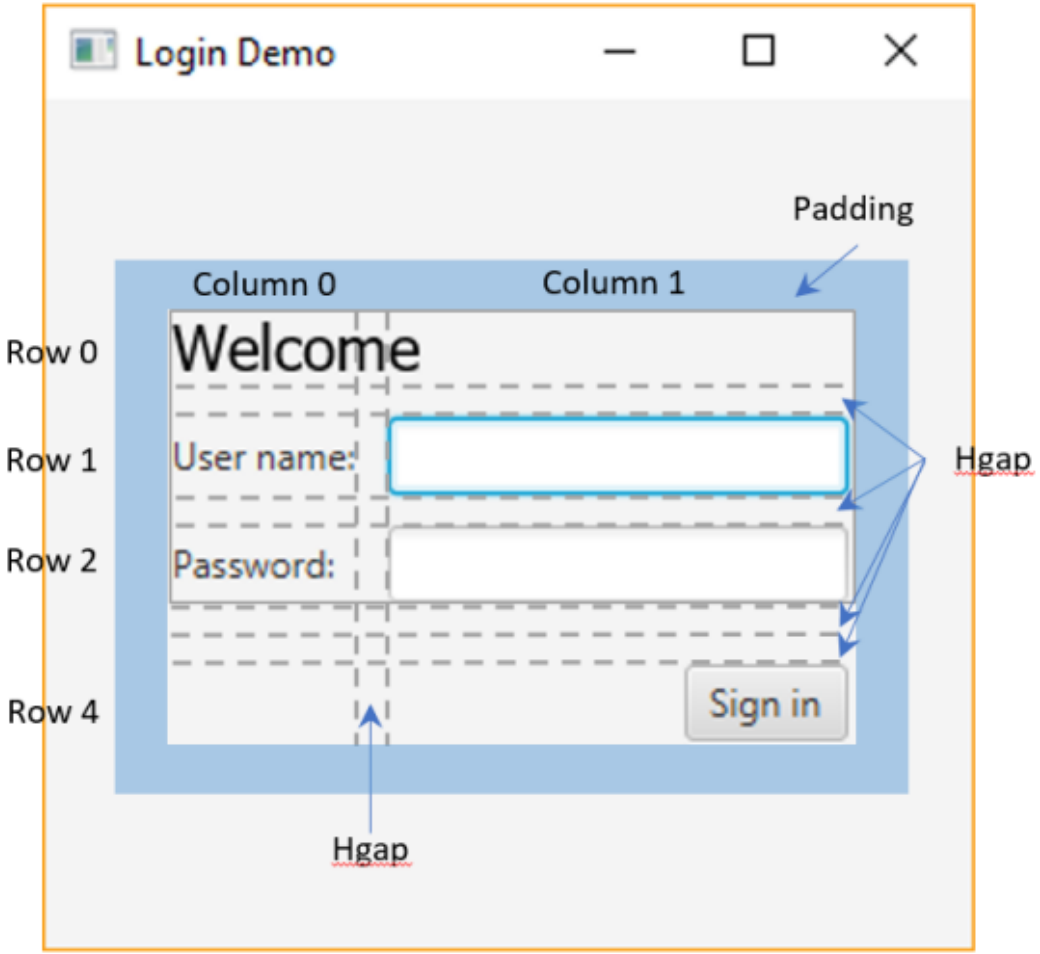
```
public class LoginPageDemo extends
  Application {
  …
}
```

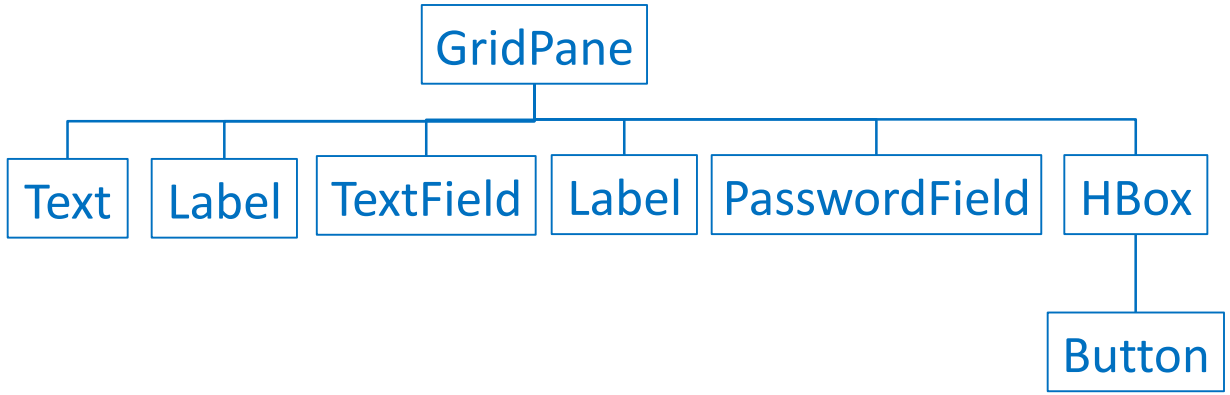
LoginPageDemo.java

Step 2: Override the start method inherited from Application

```
public class LoginPageDemo extends Application {
  @Override
  public void start(Stage stage) {
    …
  }
```

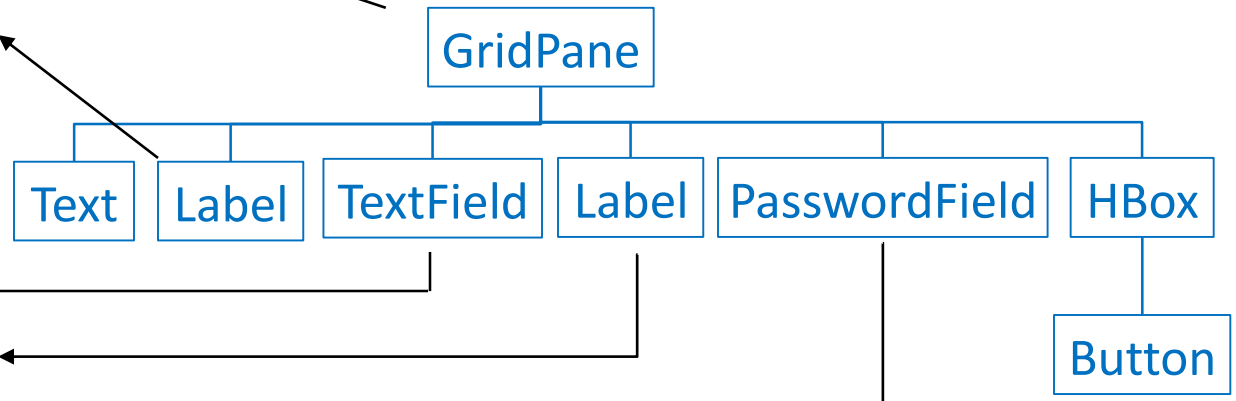
Step 3a: Prepare a Scene Graph





Step 3a: Prepare a Scene Graph

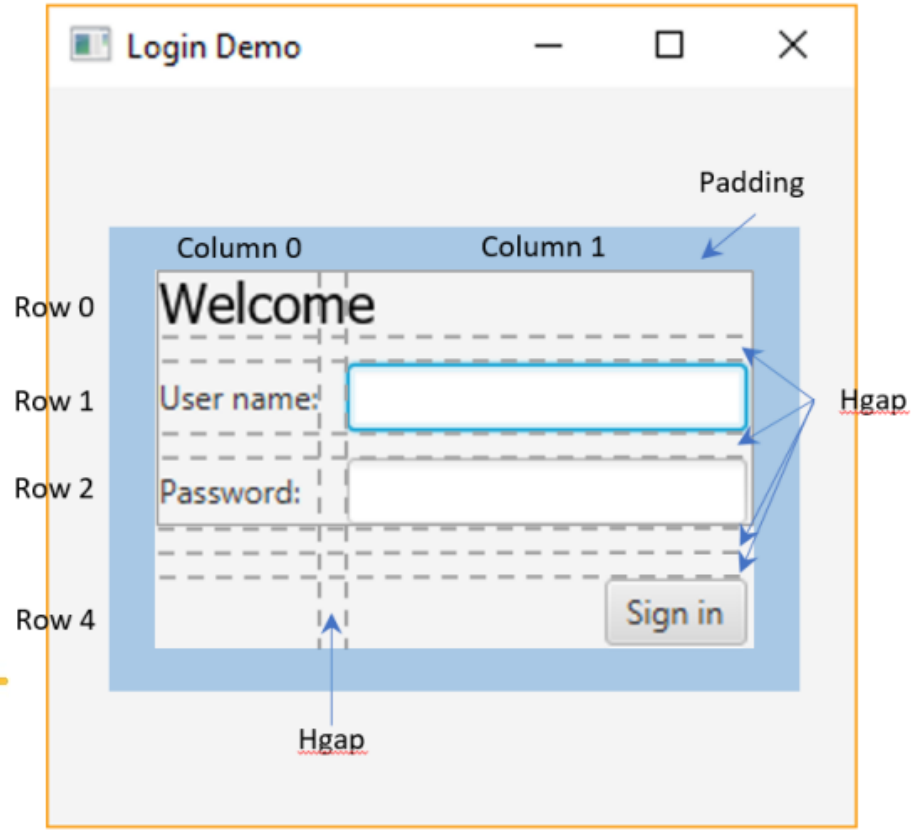
```
public class LoginPageDemo extends Application {
  @Override
  public void start(Stage stage) {
    var grid = new GridPane(); // Prepare a Scene Graph, user grid as root
    grid.setAlignment(Pos.CENTER);
    …
    var userName = new Label("User name:");
    grid.add(userName, 0, 1);
    TextField userTextField = new TextField();
    grid.add(userTextField, 1, 1);

    var password = new Label("Password:");
    grid.add(password, 0, 2);
    PasswordField passwordField = new PasswordField();
    grid.add(passwordField, 1, 2);
    …
  }
}
```



LoginPageDemo.java

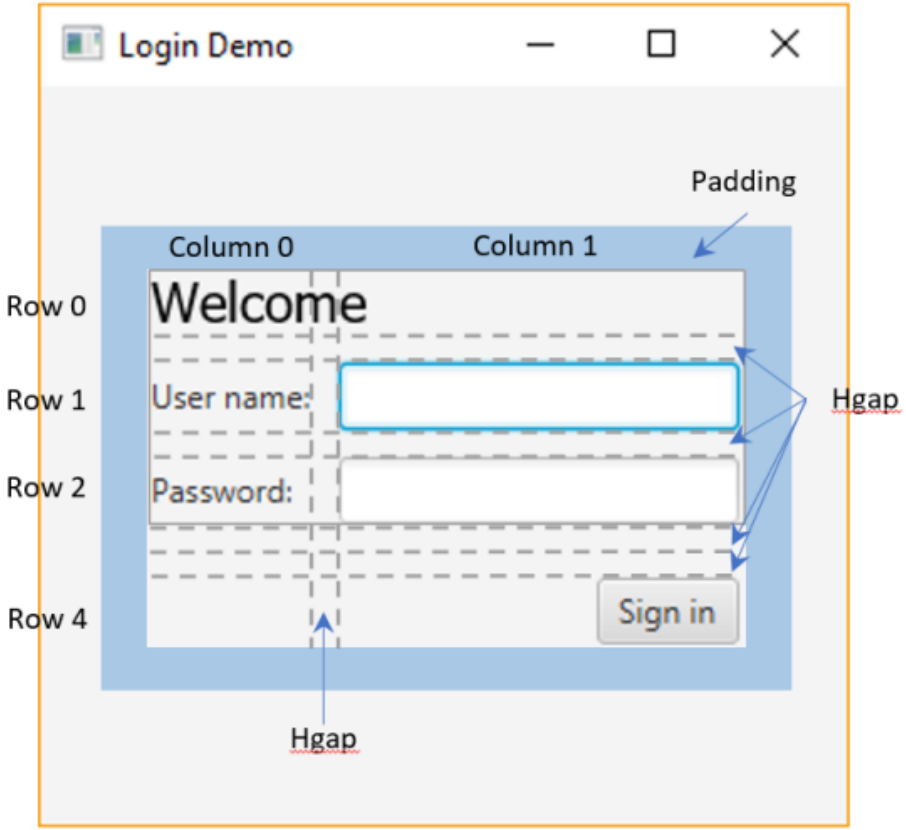
Step 3b: Construct a Scene

```
public class LoginPageDemo extends Application {
  @Override
  public void start(Stage stage) {
    GridPane grid = new GridPane();
    grid.setAlignment(Pos.CENTER);
    ...
    Label password = new Label("Password:");
    grid.add(password, 0, 2);
    PasswordField passwordField = new PasswordField();
    grid.add(passwordField, 1, 2);
    ...
    // Construct a scene, with grid as the root node
    Scene scene = new Scene(grid, 300, 275);
    ...
  }
}
```



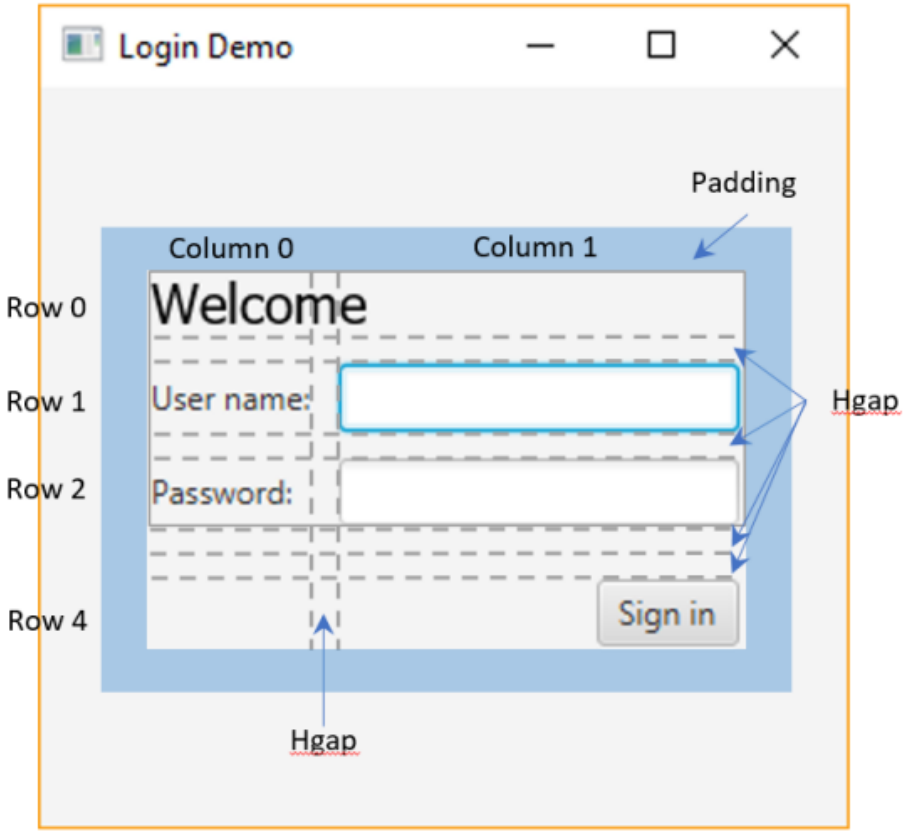
Step 3c: Set up the Stage

```
public class LoginPageDemo extends Application {
  @Override
  public void start(Stage stage) {
    GridPane grid = new GridPane();
    grid.setAlignment(Pos.CENTER);
    ...
     // Construct a scene, with grid as the root node
    Scene scene = new Scene(grid, 300, 275);
    stage.setTitle("Login Demo");
    stage.setScene(scene);
    stage.show();
  }
}
```



Step 4: Define a main Method

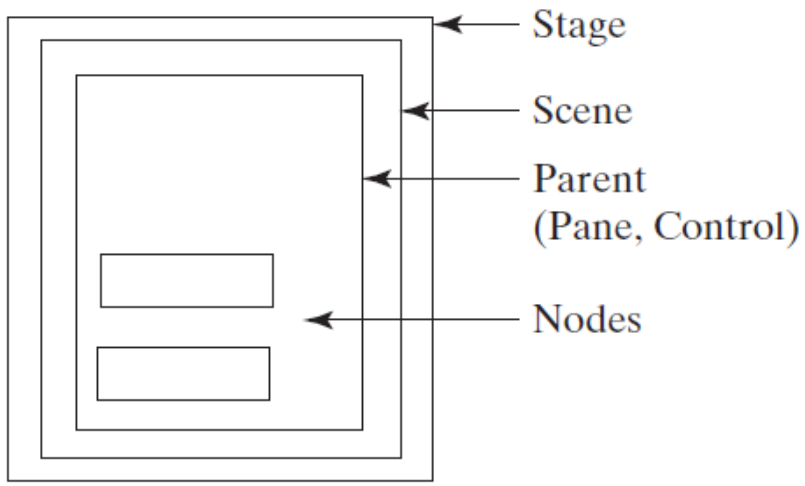
```
public class LoginPageDemo extends Application {
  @Override
  public void start(Stage stage) {
    var grid = new GridPane();
    grid.setAlignment(Pos.CENTER);
    ...
     // Construct a scene, with grid as the root node
    var scene = new Scene(grid, 300, 275);
    stage.setTitle("Login Demo");
    stage.setScene(scene);
    stage.show();
  }
  public static void main(String[] args) { launch(args); }
}
```



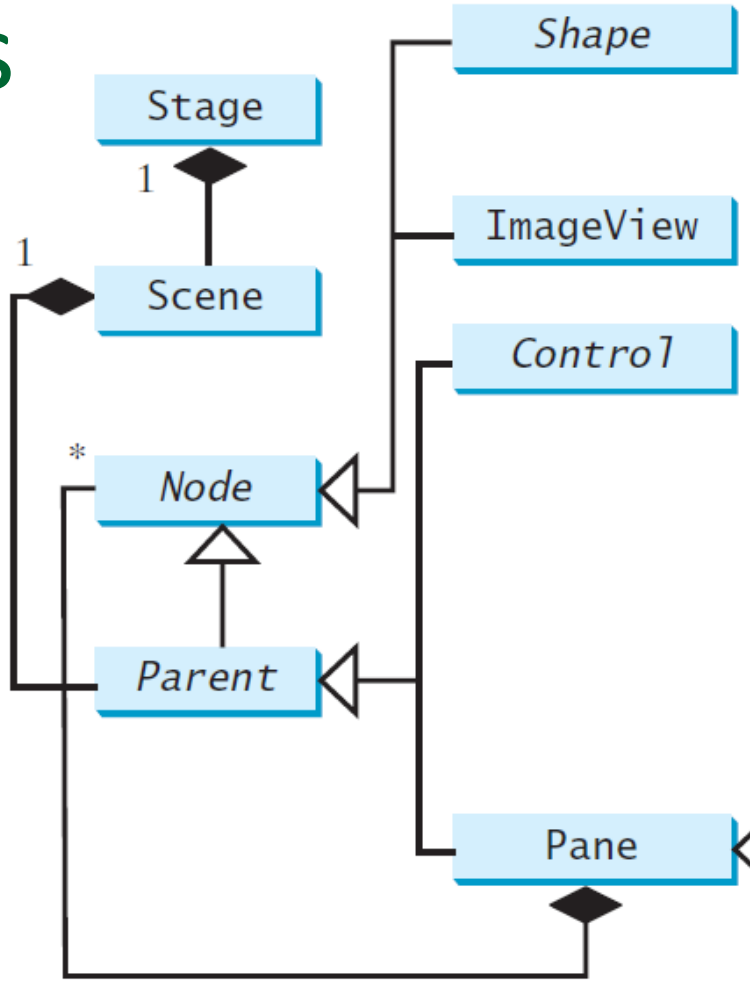
LoginPageDemo.java

Layout Panes, Shapes and UI Controls

Talk more on them in the topic of event handling later



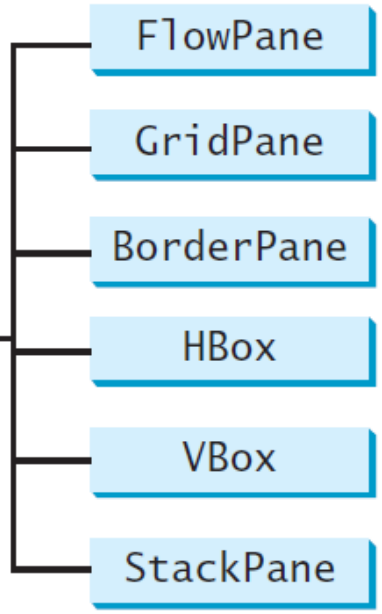
(a)



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

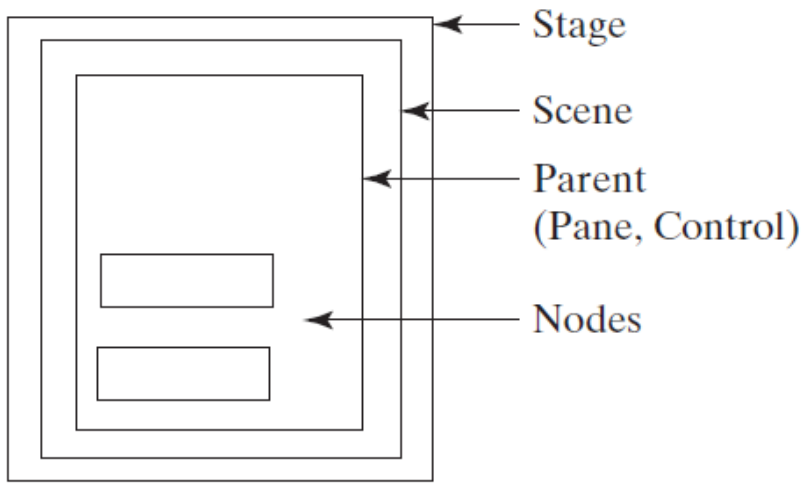
For displaying an image.

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

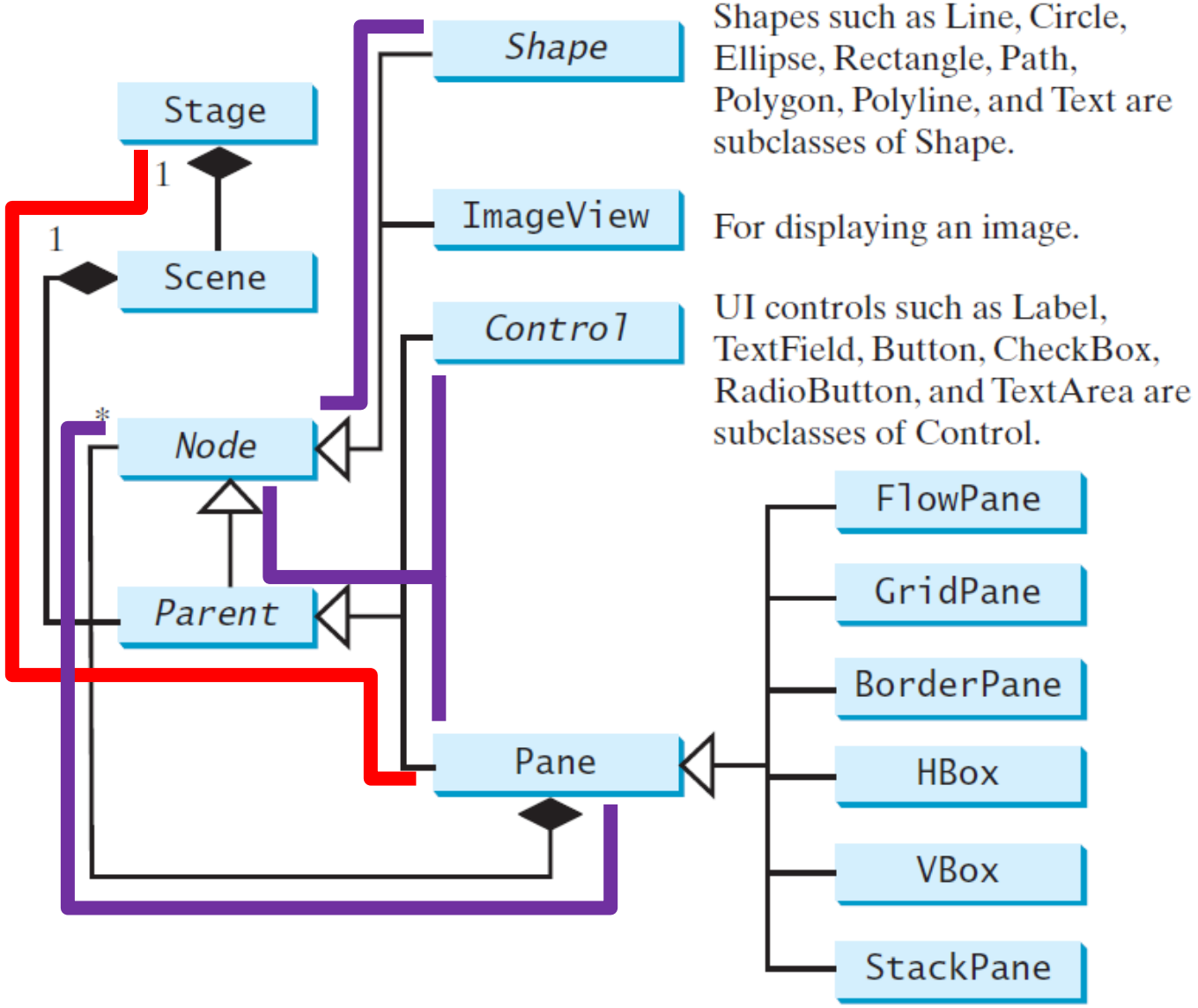


(b)

- A stage contains a scene
- A scene contains a pane
- A pane may contain multiple shapes, UI controls and other panes



(a)



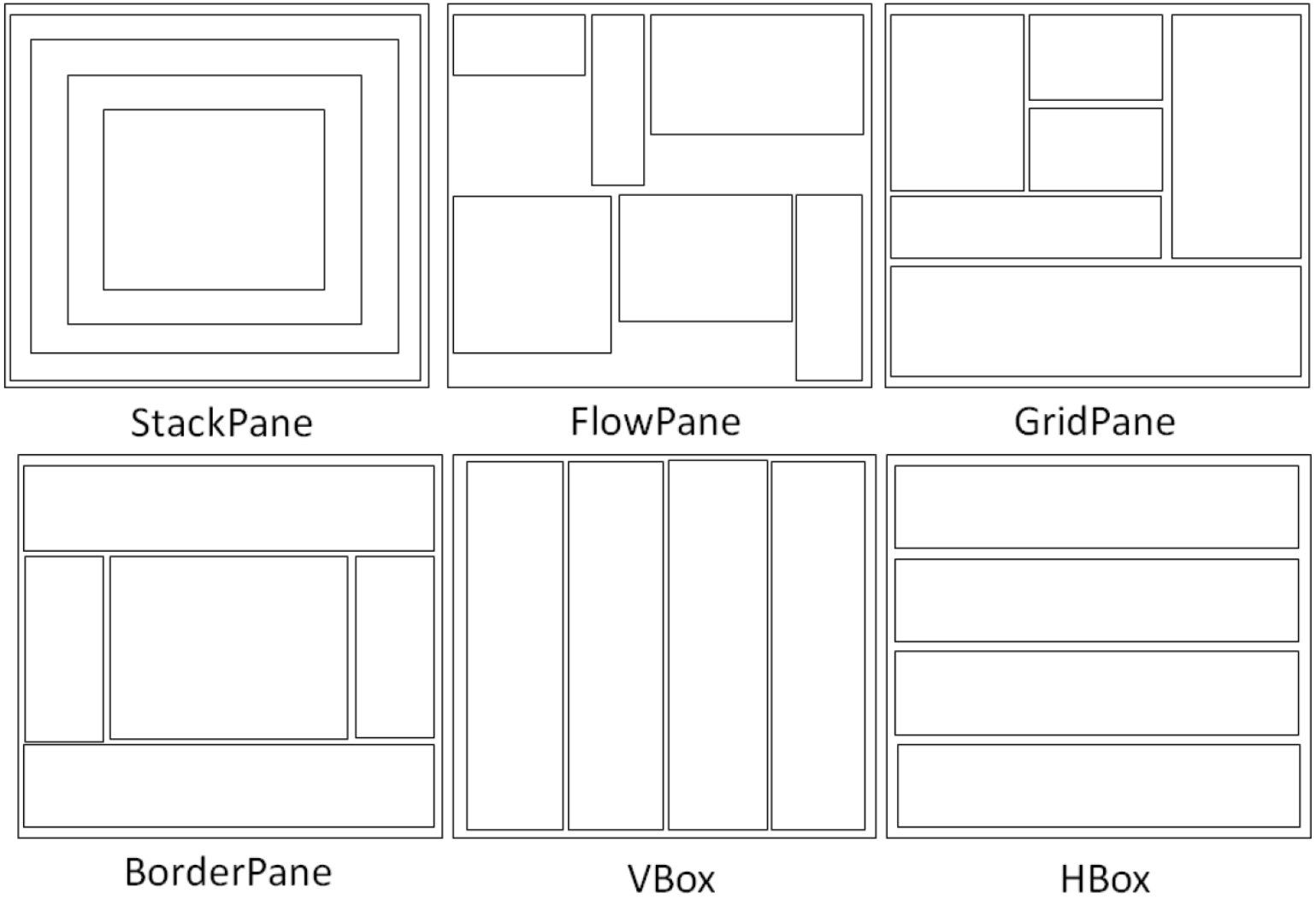
(b)

Commonly Used Layout Panes

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

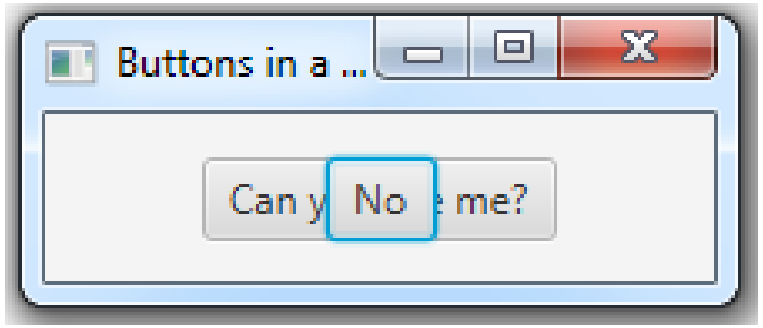
Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	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.

Commonly Used Layout Panes



StackPane

```
public void start(Stage primaryStage) {
  var stackPane = new StackPane(); // Create a scene
  stackPane.getChildren().add(new Button("Can you see me?"));
  stackPane.getChildren().add(new Button("No"));
  var scene = new Scene(stackPane, 200, 50);
  primaryStage.setTitle("Buttons in a stackpane"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
}

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



ShowStackPane.java

FlowPane

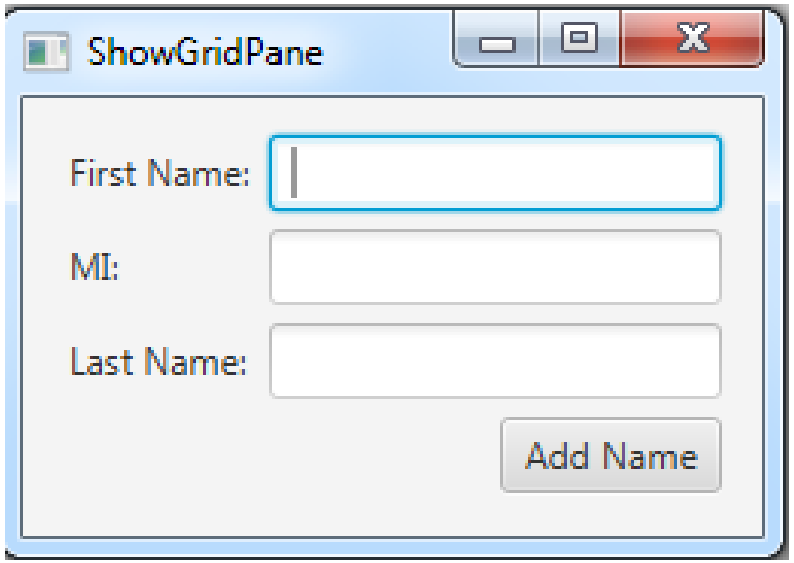

```
public void start(Stage primaryStage) {
  var pane = new FlowPane(); // Create a pane and set its properties
  ...
  // Place nodes in the pane
  pane.getChildren().addAll(new Label("First Name:"), new TextField(), new Label("MI:"));
  var tfMi = new TextField();
  tfMi.setPrefColumnCount(1); // set the width to one column
  pane.getChildren().addAll(tfMi, new Label("Last Name:"), new TextField());

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

ShowFlowPane.java

GridPane

```
public void start(Stage primaryStage) {
  var pane = new GridPane(); // Create a pane and set its properties
  pane.setAlignment(Pos.CENTER);
  …
  pane.add(new Label("First Name:"), 0, 0); // Place nodes in the pane
  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);
  var btAdd = new Button("Add Name");
  pane.add(btAdd, 1, 3);
  GridPane.setHalignment(btAdd, HPos.RIGHT);
  var scene = new Scene(pane); // Create a scene
  primaryStage.setTitle("ShowGridPane"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
}
```

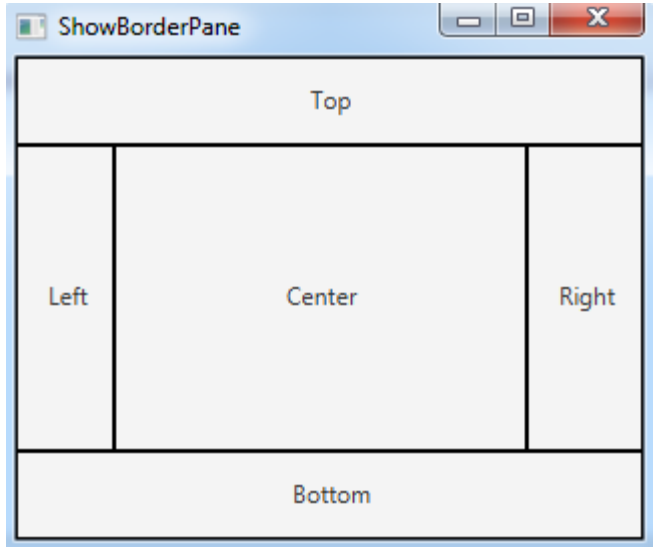
column row



ShowGridPane.java

BorderPane

```
public void start(Stage primaryStage) {
  BorderPane pane = new BorderPane(); // Create a border pane
  pane.setTop(new CustomPane("Top")); // Place nodes in the pane
  pane.setRight(new CustomPane("Right"));
  pane.setBottom(new CustomPane("Bottom"));
  pane.setLeft(new CustomPane("Left"));
  pane.setCenter(new CustomPane("Center"));
  Scene scene = new Scene(pane); // Create a scene
  ...
 }
```

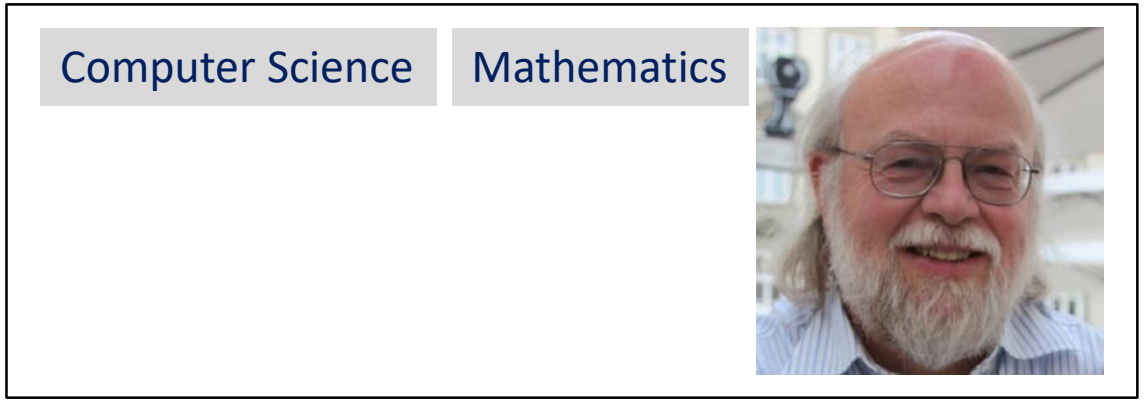



```
// Define a custom pane to hold a label in the center of the pane
class CustomPane extends StackPane {
 public CustomPane(String title) { getChildren().add(new Label(title)); }
}
```

ShowBorderPane.java

Hbox

```
// Define a method to return an HBox with
// two buttons and an image
private HBox getHBox() {
  var 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("Mathematics"));
  ImageView imageView = new ImageView(new Image("james.jpg"));
  hBox.getChildren().add(imageView);
  return hBox;
}
```



ShowHBoxVBox.java

VBox

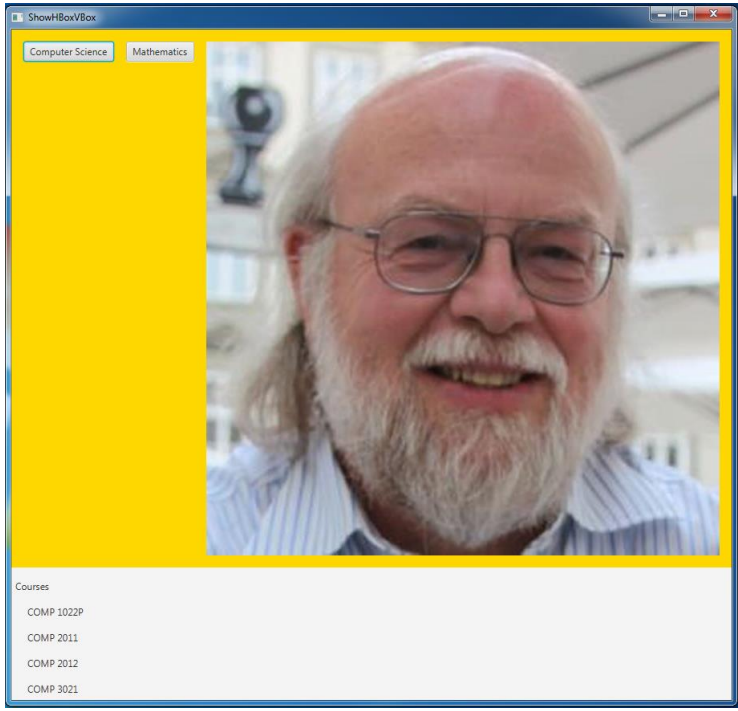
```
// Define a method to return a VBox with five labels
private VBox getVBox() {
  var vBox = new VBox(15);
  …
  vBox.getChildren().add(new Label("Courses"));
  Label[] courses = {new Label("COMP 1022P"),
                     new Label("COMP 2011"),
                     new Label("COMP 2012"),
                     new Label("COMP 3021")};
  for (var course: courses) {
    vBox.getChildren().add(course);
  }
  return vBox;
}
```

Courses
COMP 1022P
COMP 2011
COMP 2012
COMP 3021

ShowHBoxVBox.java

Adding HBox and VBox to a BorderPane

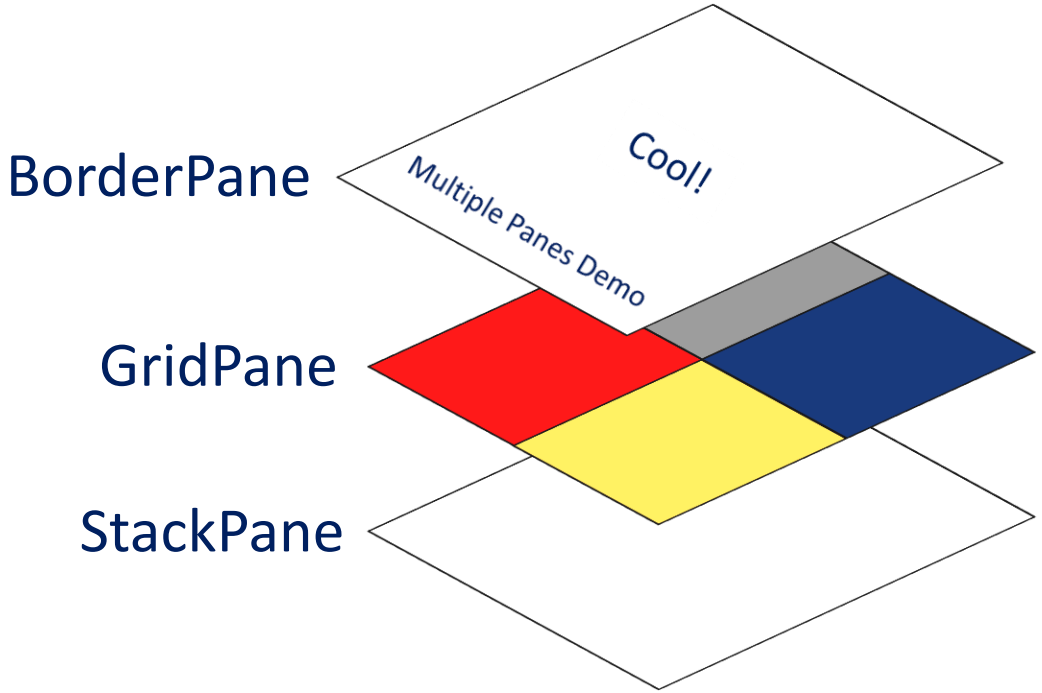
```
public void start(Stage primaryStage) {
  var pane = new BorderPane(); // Create a border pane
  pane.setTop(getHBox()); // Place nodes in the pane
  pane.setLeft(getVBox());
  var scene = new Scene(pane); // Create a scene
  primaryStage.setTitle("ShowHBoxVBox"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
}
```



ShowHBoxVBox.java

Using Multiple Panes in an Application

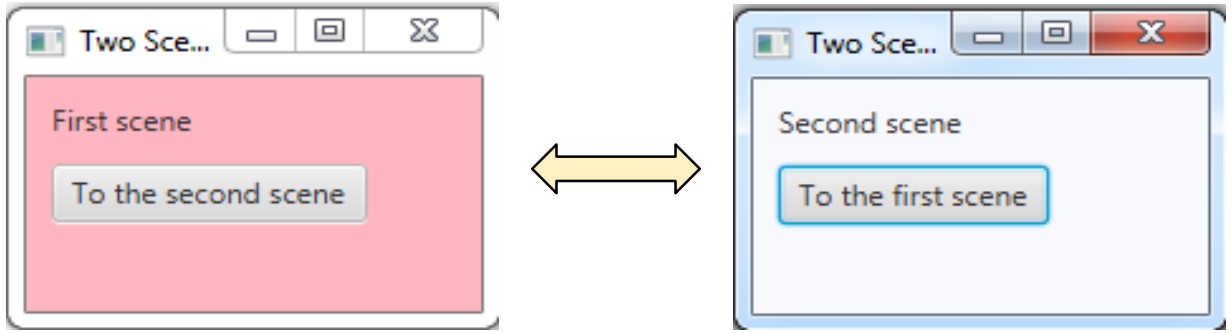
```
public void start(Stage stage) {
  …
  var grid = new GridPane();
  …
  var borderPane = new BorderPane();
  …
  var stackPane = new StackPane();
  stackPane.getChildren().add(grid);
  stackPane.getChildren().add(borderPane);
  stage.setTitle("Multiple Panes Demo");
  stage.setScene(new Scene(stackPane, 300, 300));
  stage.show();
}
```



MultiplePanesDemo.java

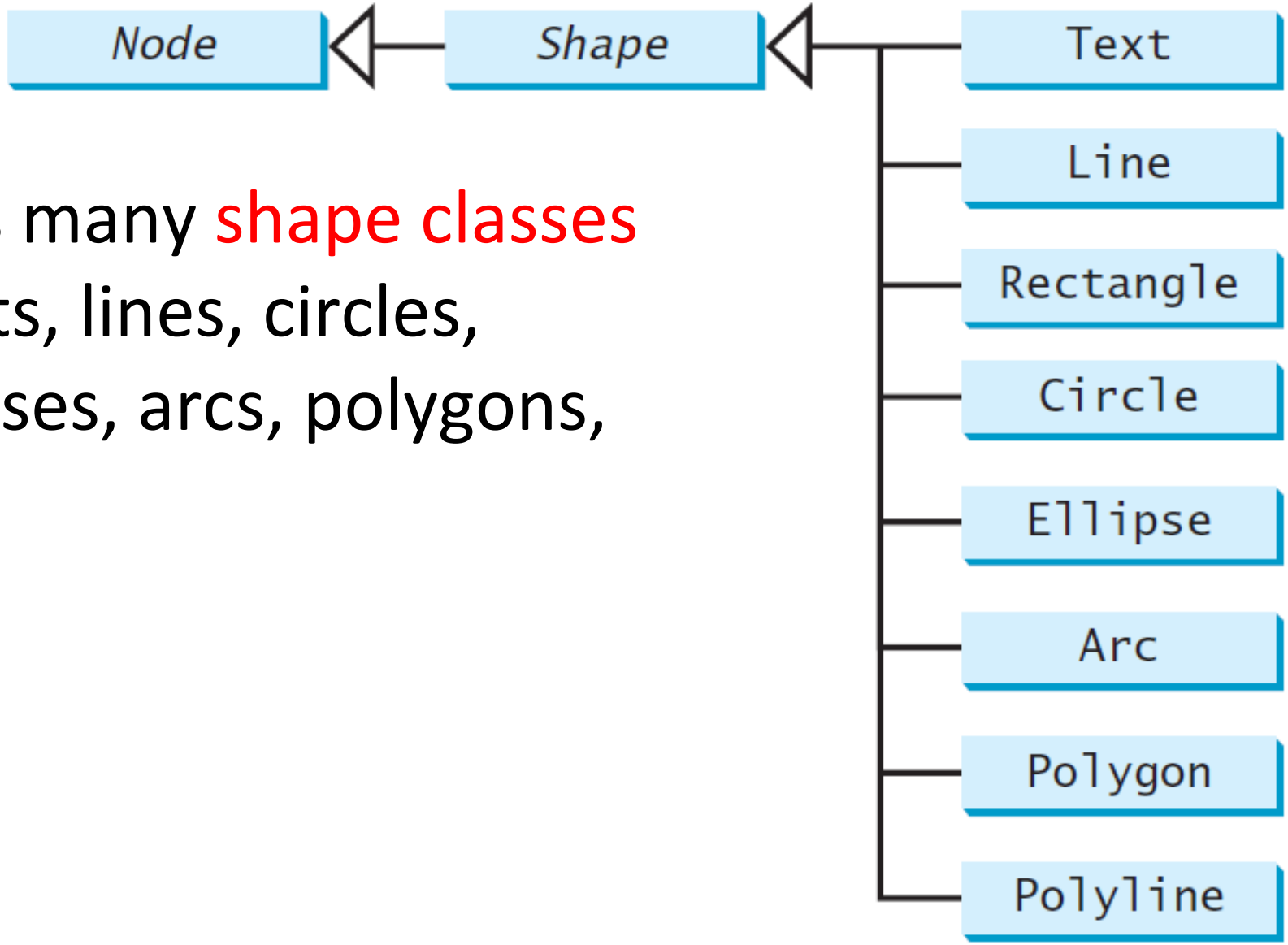
Multiple Scenes in an Application

```
@Override
public void start(Stage primaryStage) {
  theStage = primaryStage;
  …
  pane1.getChildren().addAll(labelScene1, buttonScene1);
  pane2.getChildren().addAll(labelScene2, buttonScene2);
  scene1 = new Scene(pane1, 180, 100);
  scene2 = new Scene(pane2, 180, 100);
  …
  primaryStage.setScene(scene1);
  primaryStage.show();
}
public void buttonClicked(ActionEvent e) {
  theStage.setScene( (e.getSource() == buttonScene1) ? scene2 : scene1 );
}
```



TwoSceneDemo.java

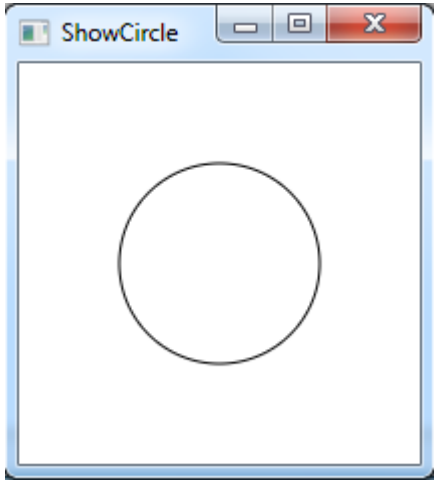
Shapes

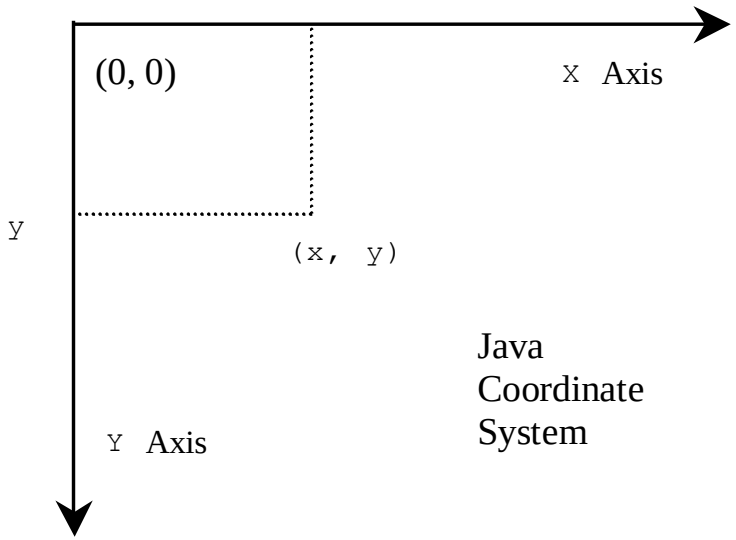


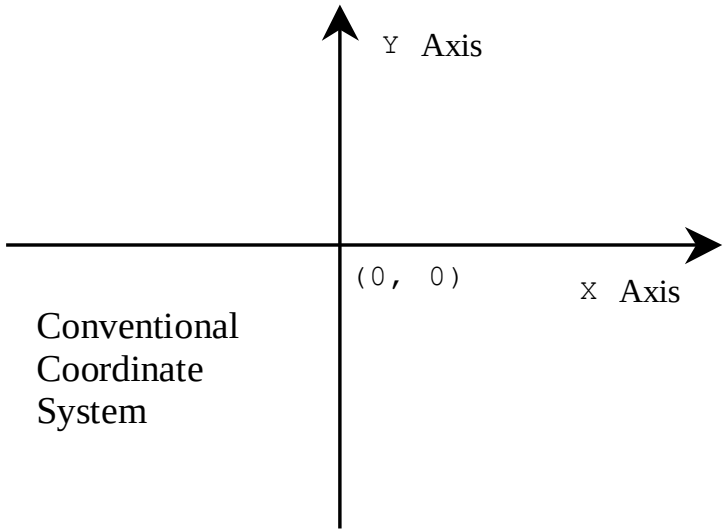
- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.

JavaFX Coordinate System

- The origin (0, 0) is at the top-left corner
- x-coordinate value goes from left to right
- y-coordinate value goes from top to bottom



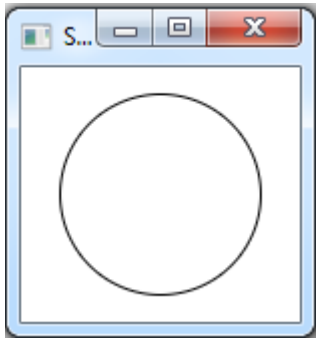


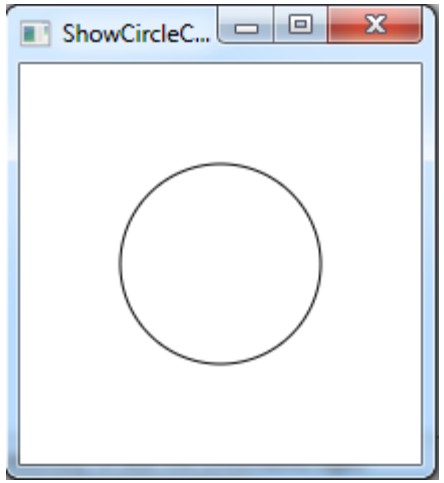


ShowCircle.java

Binding Properties

- JavaFX introduces a new concept called binding property that enables a target object to be bound to a source object
- If the source object's value changes, the target property also changes automatically.
- The target object is simply called a binding object or a binding property. For example:
  - Source object: pane.widthProperty().divide(2)
  - Target object / binding property: circle's centerX





ShowCircleCentered.java

Keeping Circle in the Center of a Pane

```
public void start(Stage primaryStage) {
  // Create a pane to hold the circle
  var pane = new Pane();
  // Create a circle and set its properties
  var circle = new Circle();
  circle.centerXProperty().bind(pane.widthProperty().divide(2));
  circle.centerYProperty().bind(pane.heightProperty().divide(2));
  circle.setRadius(50);
  pane.getChildren().add(circle); // Add circle to the pane
  // Create a scene and place it in the stage
  var 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
}
```

ShowCircleCentered.java

Binding Property: getter, setter, and property getter

TEMPLATE

Mapping: propertyType → DoubleProperty
x → centerX

```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

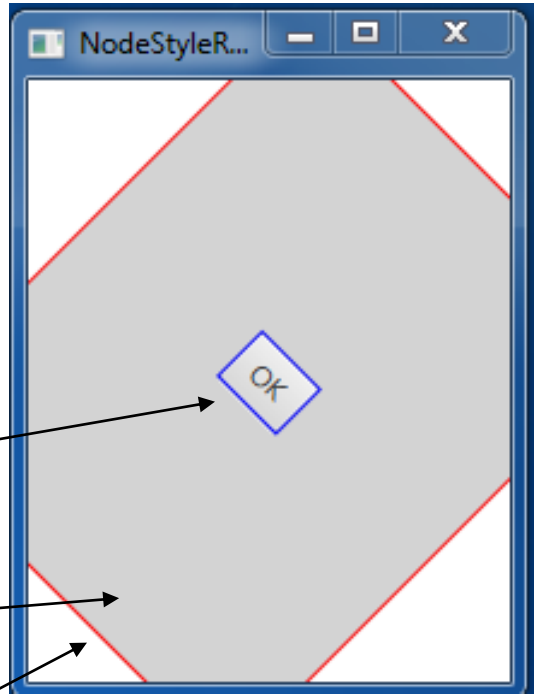
Common Properties and Methods for Nodes

- style: set a JavaFX **CSS style**
- rotate: Rotate a node

```
public void start(Stage primaryStage) {
  // Create a scene and place a button in the scene
  var pane = new StackPane();
  var 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;");
  var 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
}
```

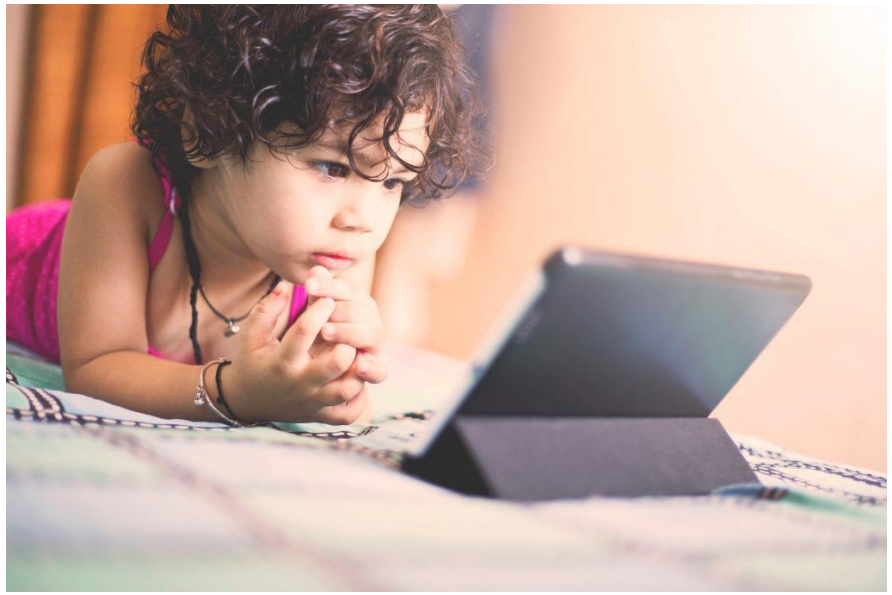
blue button border

rotate 45 degrees



NodeStyleRotateDemo.java

Selection of Commonly Used GUI Classes (for self study after 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	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 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.

ShowImage.java

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)
```

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.java

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.

FontDemo.java

Text

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.

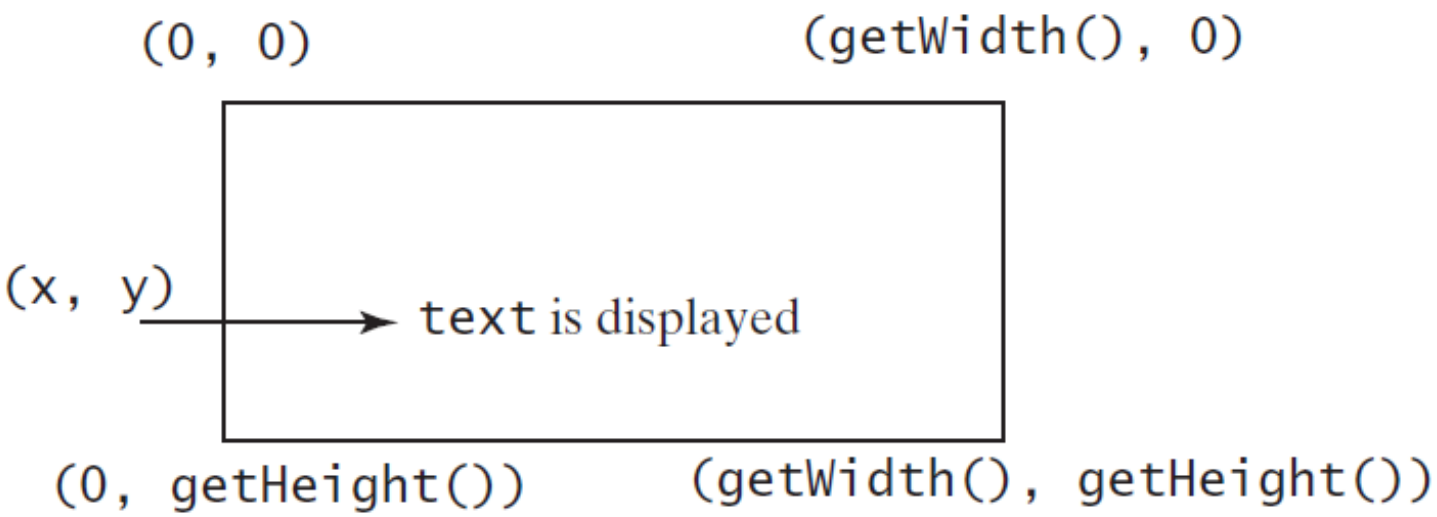
javafx.scene.text.Text

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

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

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)

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

ShowText.java

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.

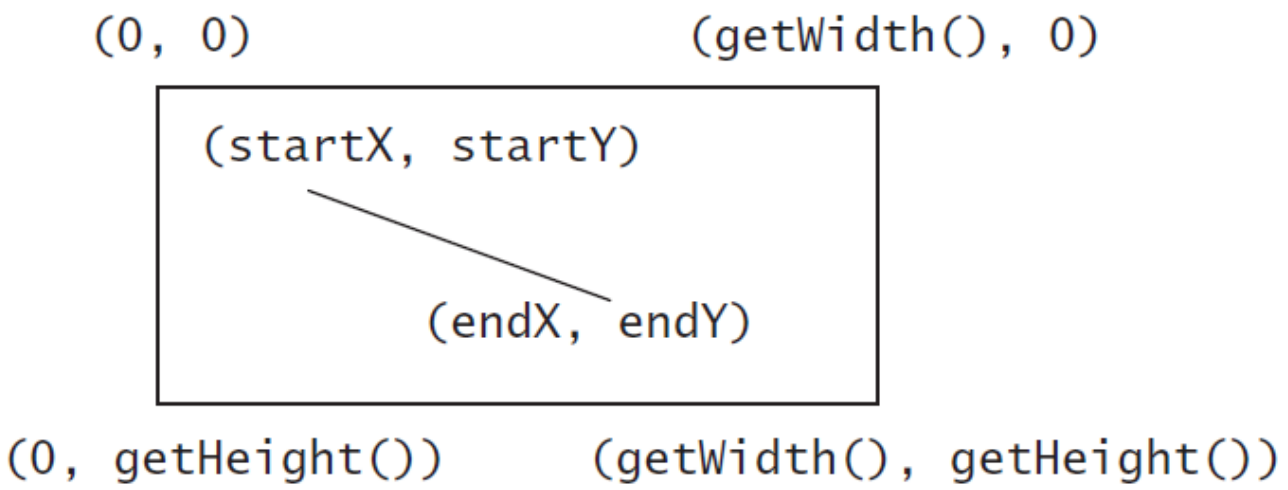
javafx.scene.shape.Line

```
-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.



ShowLine.java

Rectangle

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

javafx.scene.shape.Rectangle

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

-arcHeight: DoubleProperty
```

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

The x-coordinate of the upper-left corner of the rectangle (default 0).

The y-coordinate of the upper-left corner of the rectangle (default 0).

The width of the rectangle (default: 0).

The height of the rectangle (default: 0).

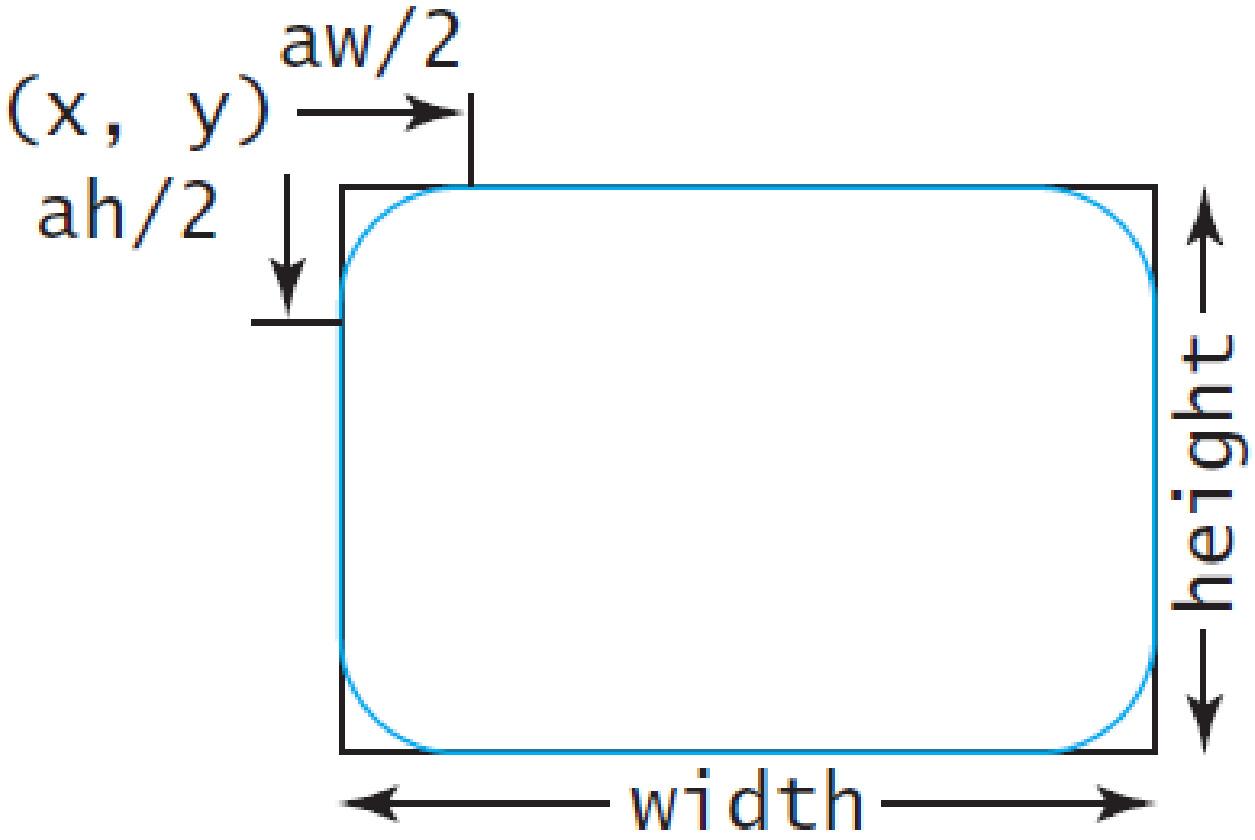
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

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

Creates an empty `Rectangle`.

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

Rectangle Example



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

ShowRectangle.java

Circle

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.Circle

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

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

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

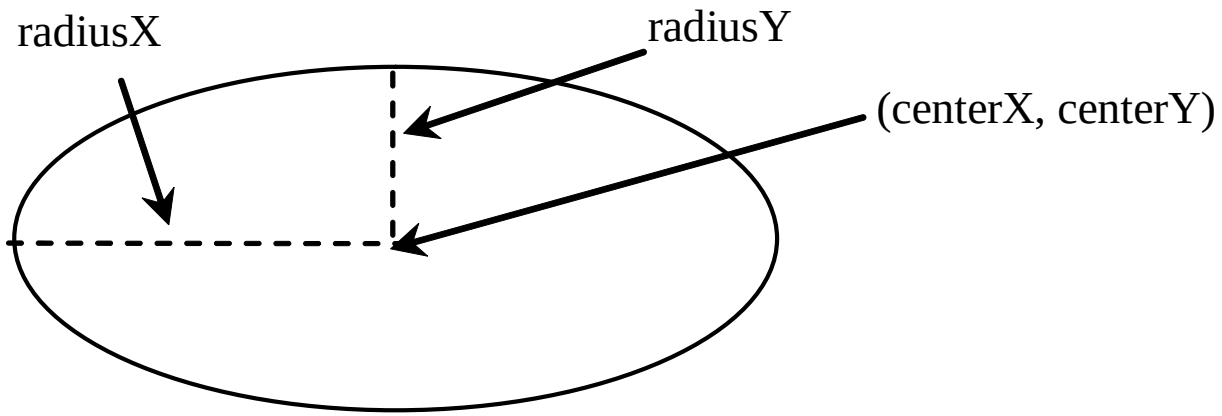
javafx.scene.shape.Ellipse

```
-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.



ShowEllipse.java

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.

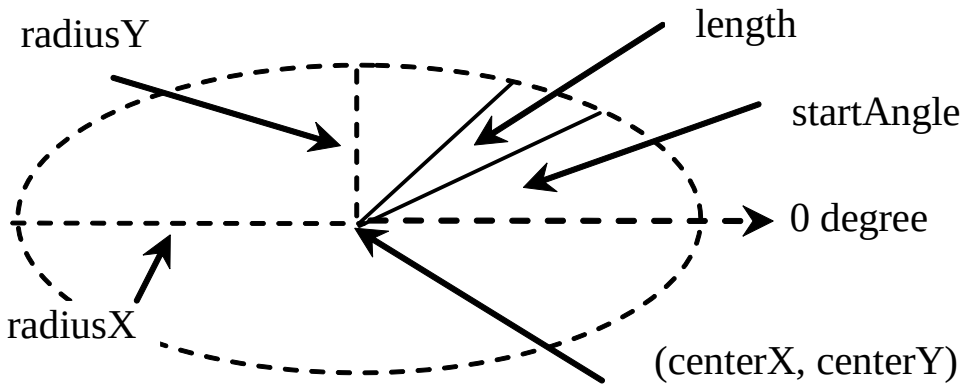
The angular extent of the arc in degrees.

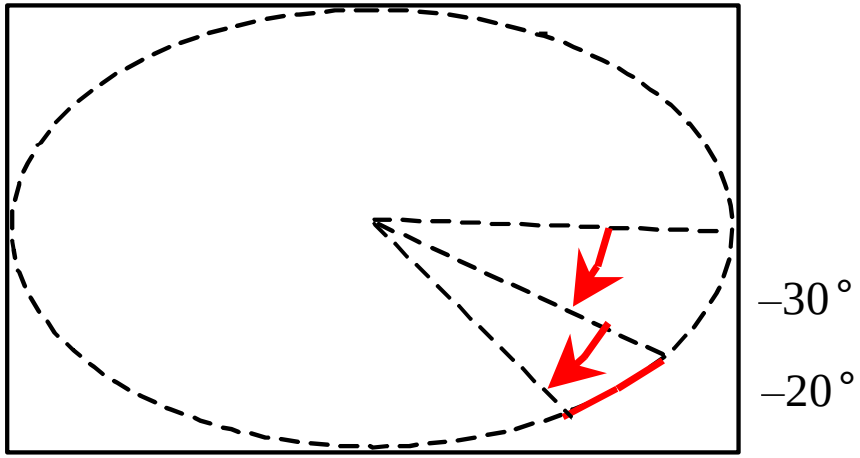
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

Creates an empty `Arc`.

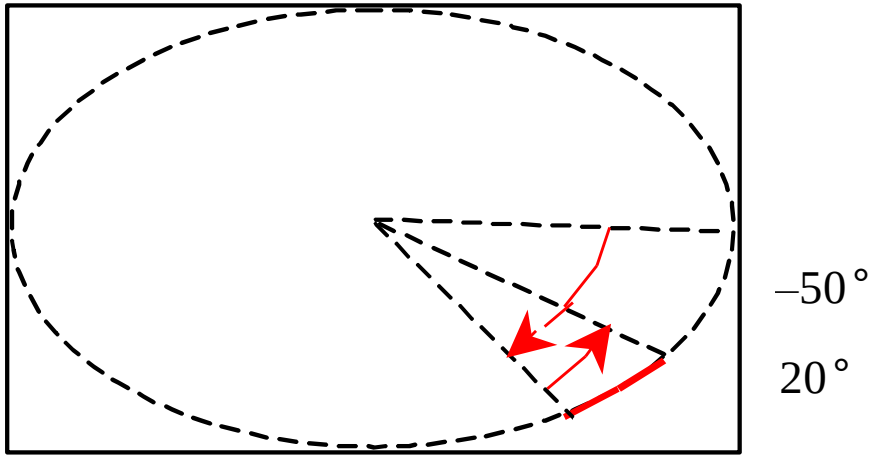
Creates an `Arc` with the specified arguments.

Arc Examples





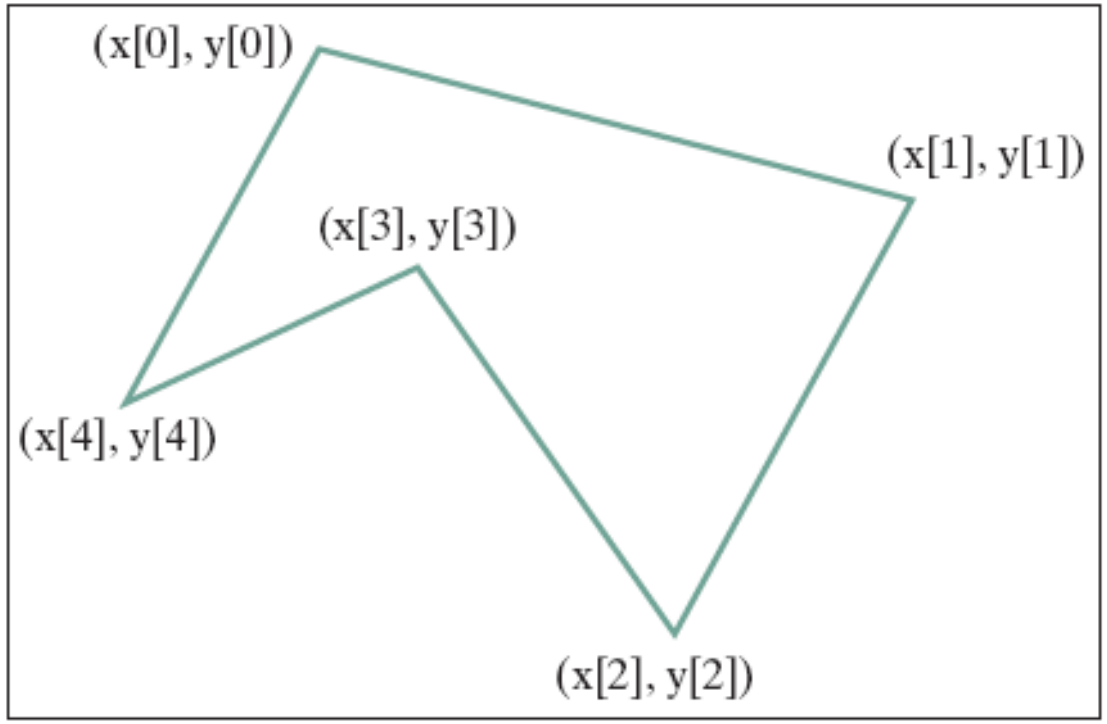
(a) Negative starting angle –30° and negative spanning angle –20°



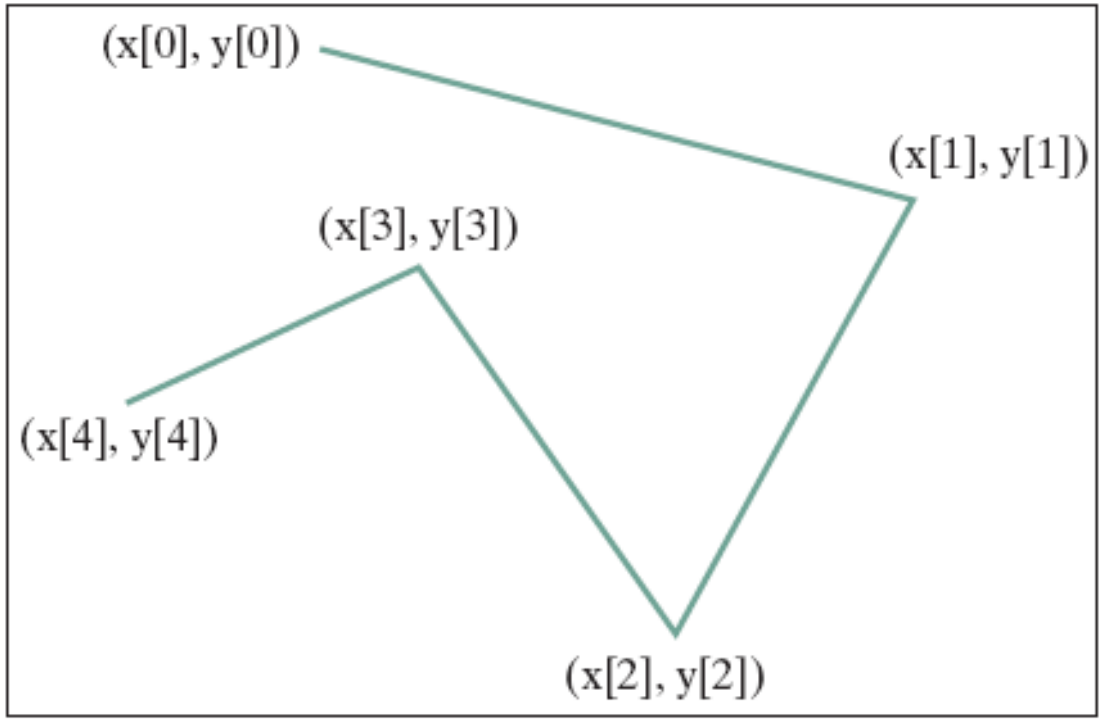
(b) Negative starting angle –50° and positive spanning angle 20°

ShowArc.java

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.java

JavaFX (latest version is JavaFX 15)

- JavaFX can be programmed by Java commands or XML commands (FXML) which Java can load.
- JavaFX is built to support common touch gestures in mobile devices.
- There is a nice JavaFX designer called SceneBuilder to create fancy layouts in FXML.
  - It separates user interface design from code.
  - Useful links:
    - https://gluonhq.com/products/scene-builder/ Download link for Scene Builder
    - https://docs.oracle.com/javafx/scenebuilder/1/use_java_ides/sb-with-intellij.htm
    - https://www.youtube.com/watch?v=Z1W4E2d4Yxo