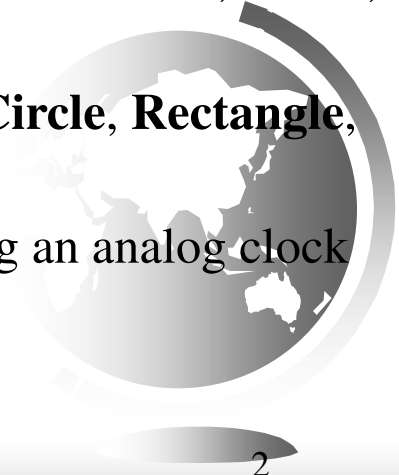


Chapter 14 JavaFX Basics



Objectives

- ☞ 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, UI controls, and shapes (§14.4).
- ☞ To use binding properties to synchronize property values (§14.5).
- ☞ To use the common properties **style** and **rotate** for nodes (§14.6).
- ☞ To create colors using the **Color** class (§14.7).
- ☞ 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).



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.



Introduction

- ☞ There are **two sets of Java APIs** for graphics programming:
 - **AWT** (Abstract Windowing Toolkit) and
 - **Swing**
- ☞ **AWT API** was introduced in JDK 1.0. Most of the AWT components have **become obsolete** and should be **replaced by newer Swing** components.
- ☞ **Swing API**, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of **Java Foundation Classes (JFC)** after the release of JDK 1.1.
- ☞ JFC consists of **Swing**, *Java2D*, *Accessibility*, *Internationalization*, and *Pluggable Look-and-Feel Support APIs*. JFC was an add-on to JDK 1.1 but has been integrated into core Java since JDK 1.2.

JavaFX

- ➡ **JavaFX** is a software platform for creating and delivering desktop applications, as well as **rich internet applications** (RIAs) that can run across a wide variety of devices.
- ➡ **JavaFX** is intended to replace **Swing** as the standard *GUI library for Java SE*, but both will be included for the foreseeable future.

*IFX is just a name, which is normally related with sound or visual effects in the **javafx** i was in the belief that the **fx** was function. ...
FIPS stands for the Federal Information Processing Standardization*

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 likely 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.

```
// Create AWT Button Example
```

```
// This java example shows how to create a Button using AWT Button class.
```

```
import java.applet.Applet;
```

```
import java.awt.Button;
```

```
/*
```

```
<applet code="CreateAWTButtonExample" width=200 height=200>
```

```
</applet>
```

```
*/
```

```
public class CreateAWTButtonExample extends Applet{
```

```
    public void init(){
```

```
        /*
```

```
        * To create a button use
```

```
        * Button() constructor.
```

```
        */
```

```
        Button button1 = new Button();
```

```
        /*
```

```
        * Set button caption or label using
```

```
        * void setLabel(String text)
```

```
        * method of AWT Button class.
```

```
        */
```

```
        button1.setLabel("My Button 1");
```

```
        /*
```

```
        * To create button with the caption use
```

```
        * Button(String text) constructor of
```

```
        * AWT Button class.
```

```
        */
```

```
        Button button2 = new Button("My Button 2");
```

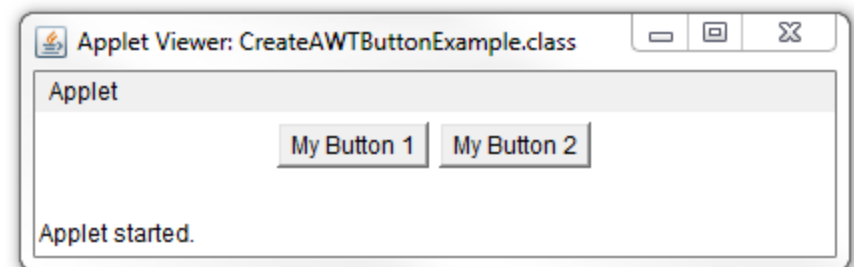
```
        //add buttons using add method
```

```
        add(button1);
```

```
        add(button2);
```

```
    }
```

```
}
```



Links for API Resources

- ☞ You need to check the JDK API specification (<http://docs.oracle.com/javase/7/docs/api/index.html>) for the AWT and Swing APIs while reading this chapter. The best online reference for Graphics programming is the "Swing Tutorial" @ <http://docs.oracle.com/javase/tutorial/uiswing/>. For advanced 2D graphics programming, read "Java 2D Tutorial" @ <http://docs.oracle.com/javase/tutorial/2d/index.html>.

Programming GUI with AWT

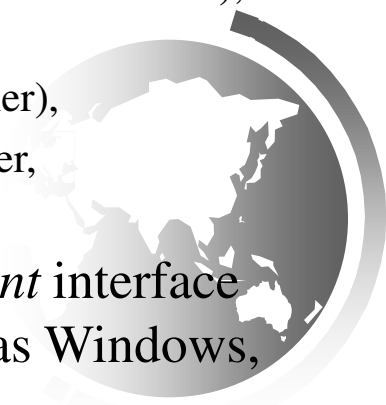
Java Graphics **APIs** - **AWT** and **Swing** - **provide a huge set of reusable GUI components**, such as:

- Button
 - text field label,
 - choice,
 - panel and
 - frame
- for building GUI applications.



AWT Packages

- ☞ **AWT is huge! It consists of 12 packages**
- ☞ **Swing is even bigger, with 18 packages as of JDK 1.7!.**
- ☞ Fortunately, only 2 packages - `java.awt` and `java.awt.event` - are commonly-used.
- ☞ The `java.awt` package contains the *core* AWT graphics classes:
 - ◆ GUI Component classes (such as `Button`, `TextField`, and `Label`),
 - ◆ GUI Container classes (such as `Frame`, `Panel`, `Dialog` and `ScrollPane`),
 - ◆ Layout managers (such as `FlowLayout`, `BorderLayout` and `GridLayout`),
 - ◆ Custom graphics classes (such as `Graphics`, `Color` and `Font`).
- ☞ **The `java.awt.event` package** supports event handling:
 - ◆ Event classes (such as `ActionEvent`, `MouseEvent`, `KeyEvent` and `WindowEvent`),
 - ◆ Event Listener Interfaces (such as `ActionListener`, `MouseListener`, `KeyListener` and `WindowListener`),
 - ◆ Event Listener Adapter classes (such as `MouseAdapter`, `KeyAdapter`, and `WindowAdapter`).
- ☞ **AWT** provides a *platform-independent* and *device-independent* interface to develop graphic programs that runs on all platforms, such as Windows, Mac, and Linux.



```

import java.awt.*;    // Using AWT container and component classes
import java.awt.event.*; // Using AWT event classes and listener interfaces
// An AWT program inherits from the top-level container java.awt.Frame
public class AWTCCounter extends Frame implements ActionListener {
    private Label lblCount;    // Declare component Label
    private TextField tfCount; // Declare component TextField
    private Button btnCount;   // Declare component Button
    private int count = 0;     // Counter's value

    /** Constructor to setup GUI components and event handling */
    public AWTCCounter () {
        setLayout(new FlowLayout());
        // "super" Frame sets its layout to FlowLayout, which arranges the components
        // from left-to-right, and flow to next row from top-to-bottom.

        lblCount = new Label("Counter"); // construct Label
        add(lblCount);                   // "super" Frame adds Label

        tfCount = new TextField("0", 10); // construct TextField
        tfCount.setEditable(false);        // set to read-only
        add(tfCount);                     // "super" Frame adds tfCount

        btnCount = new Button("Count"); // construct Button
        add(btnCount);                  // "super" Frame adds Button

        btnCount.addActionListener(this);
        // Clicking Button source fires ActionEvent
        // btnCount registers this instance as ActionEvent listener

        setTitle("AWT Counter"); // "super" Frame sets title
        setSize(250, 100);      // "super" Frame sets initial window size

        // System.out.println(this);
        // System.out.println(lblCount);
        // System.out.println(tfCount);
        // System.out.println(btnCount);

        setVisible(true);        // "super" Frame shows

        // System.out.println(this);
        // System.out.println(lblCount);
        // System.out.println(tfCount);
        // System.out.println(btnCount);
    }
}

```

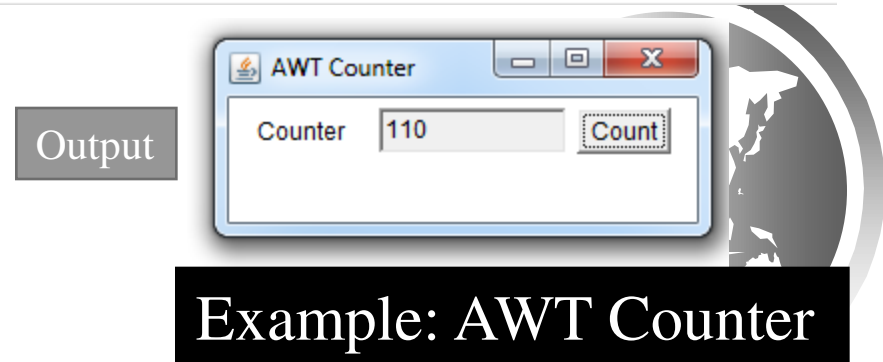
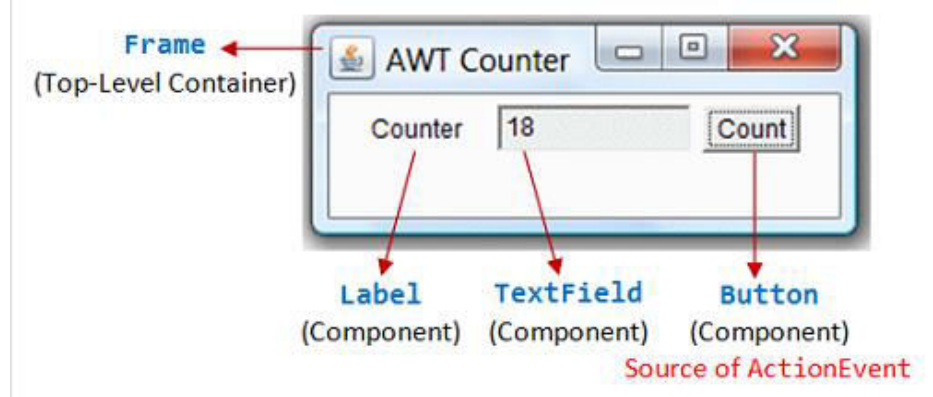
```

/** The entry main() method */
public static void main(String[] args) {
    // Invoke the constructor to setup the GUI, by allocating an
    // instance
    AWTCCounter app = new AWTCCounter();
}

/** ActionEvent handler - Called back upon button-click. */

public void actionPerformed(ActionEvent evt) {
    ++count; // increase the counter value
    // Display the counter value on the TextField tfCount
    tfCount.setText(count + ""); // convert int to String
}

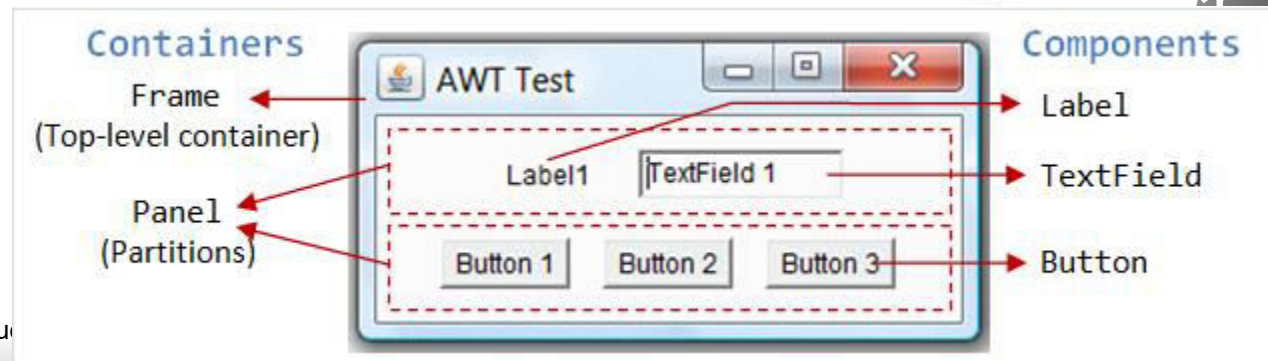
```



Example: AWT Counter

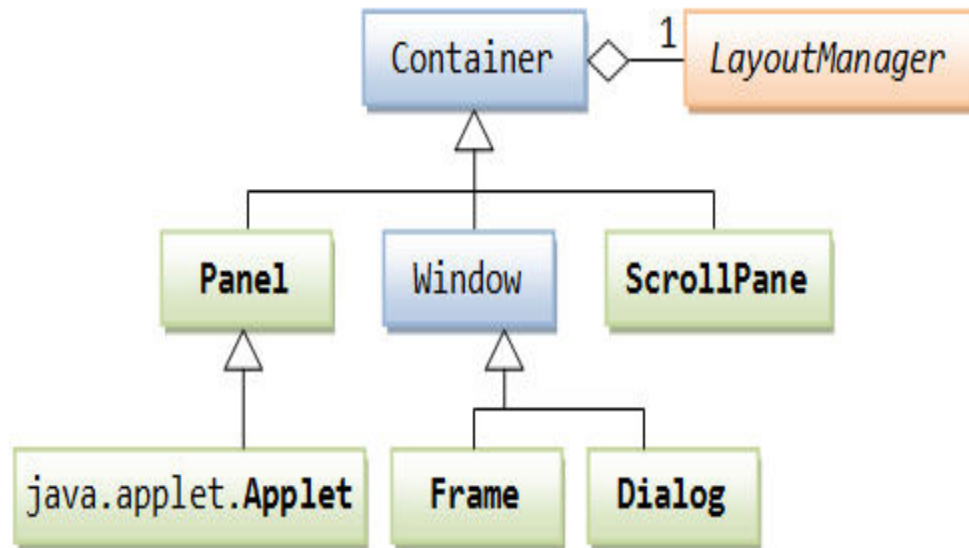
Containers and Components

- ☞ There are **two types of GUI elements**:
- **Component**: Components are elementary GUI entities (such as Button, Label, and TextField.)
 - **Container**: Containers (such as Frame, Panel and Applet) are used to *hold components in a specific layout* (such as flow or grid). A container can also hold sub-containers.
 - **GUI components are also called *controls* (Microsoft ActiveX Control), *widgets* (Eclipse's Standard Widget Toolkit, Google Web Toolkit),** which allow users to interact with (i.e., control) the application through these components (*such as button-click and text-entry*).



Hierarchy of the AWT Container Classes

The hierarchy of the AWT Container classes is as follows:



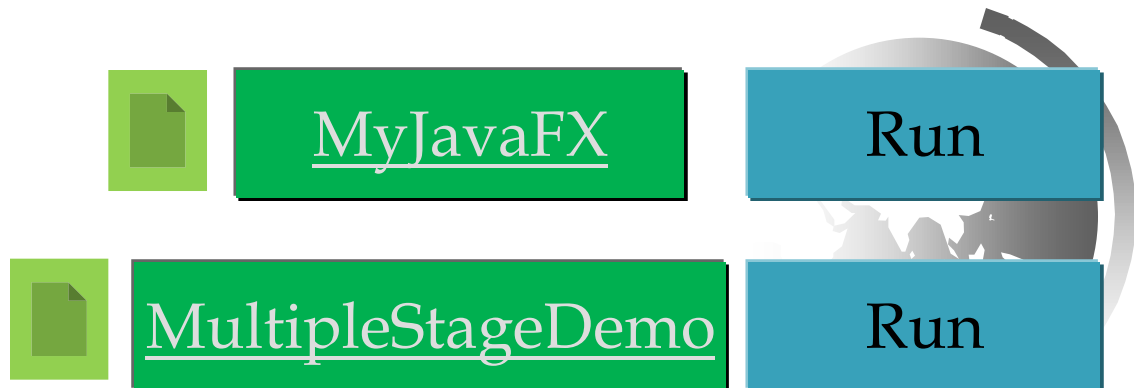
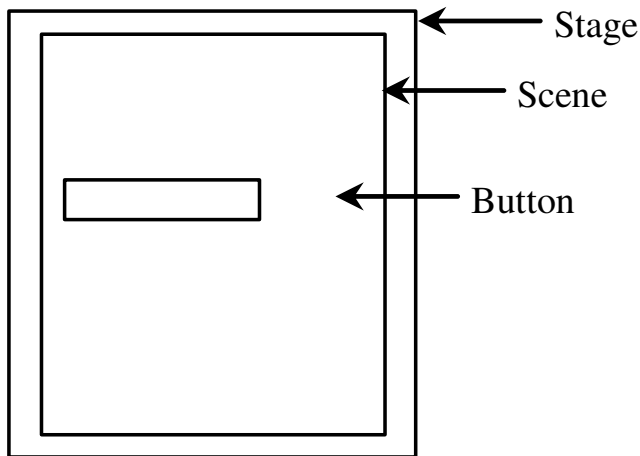
As illustrated, each Container has a layout.

http://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html



Basic Structure of JavaFX

- ➡ Application
- ➡ Override the start(Stage) method
- ➡ Stage, Scene, and Nodes



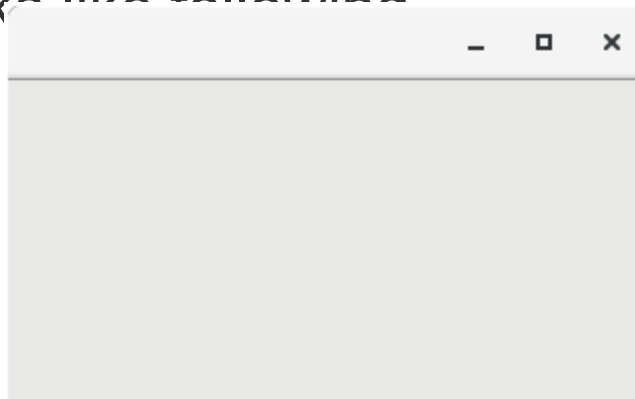
Basic Structure of JavaFX

- ➡ JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes. We need to import **javafx.application.Application** class in every JavaFX application. This provides the following life cycle methods for JavaFX application.
 - public void init()
 - public abstract void start(Stage primaryStage)
 - public void stop()
- ➡ in order to create a basic JavaFX application, we need to:
 1. Import **javafx.application.Application** into our code.
 2. Inherit **Application** into our class.
 3. Override **start()** method of Application class.



Basic Structure of JavaFX

- ➡ **Stage:**
- ➡ **Stage** in a JavaFX application is similar to the **Frame** in a Swing Application.
- ➡ It acts like a container for all the JavaFX objects. Primary Stage is created internally by the platform. Other stages can further be created by the application.
- ➡ The object of primary stage is passed to **start** method.
- ➡ We need to call **show** method on the **primary stage object** in order to show our primary stage. Initially, the primary Stage looks like following



Basic Structure of JavaFX

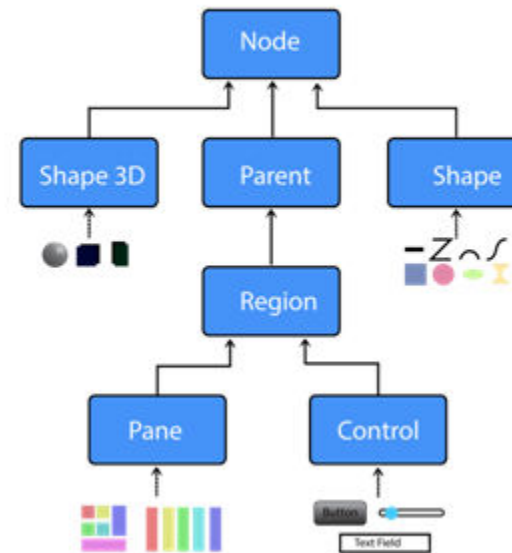
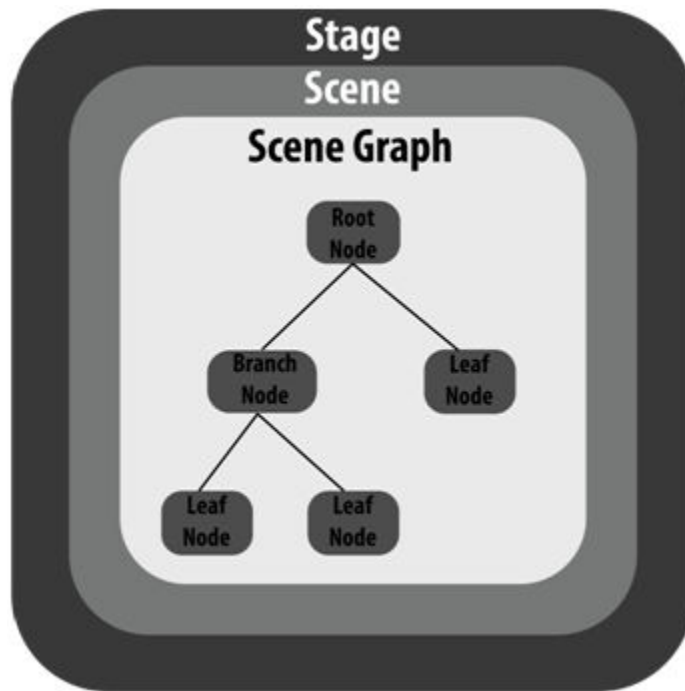
➤ **Scene:**

- Scene actually holds all the physical contents (nodes) of a JavaFX application. **Javafx.scene.Scene** class provides all the methods to deal with a scene object. Creating scene is necessary in order to visualize the contents on the stage.

➤ **Scene Graph:**

- Scene Graph exists at the lowest level of the hierarchy. It can be seen as the collection of various **nodes**. A **node** is the element which is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.
- The nodes are implemented in a tree kind of structure. There is always one root in the scene graph.

Basic Structure of JavaFX



First JavaFX Application

A simple JavaFX application which prints **hello world** on the console on clicking the button shown on the stage:

Step1: Extend `javafx.application.Application` and override `start()`

☞ **start()** method is the starting point of constructing a JavaFX application therefore we need to first override start method of `javafx.application.Application` class.

☞ Object of the class `javafx.stage.Stage` is passed into the **start()** method therefore import this class and pass its object into start method.

☞ `JavaFX.application.Application` needs to be imported in order to override start method.

```
1.package application;
2.import javafx.application.Application;
3.import javafx.stage.Stage;
4.public class Hello_World extends Application{
5.
6.    @Override
7.    public void start(Stage primaryStage)
    throws Exception {
8.                                     // TODO Auto-
    generated method stub
9.
10.    }
11.
12.}
```



Step 2: Create a Button

- ➡ A button can be created by instantiating the **javafx.scene.control.Button** class. we have to import this class into our code. Pass the button label text in Button class

constructor.

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.control.Button;
4.import javafx.stage.Stage;
5.public class Hello_World extends Application{
6.    public void start(Stage primaryStage) throws Exception {
7.        Button btn1=new Button("Say, Hello World");
8.    }
9.}
```

Step 3: Create a layout and add button to it

- ☞ JavaFX provides the number of layouts. We need to implement one of them in order to visualize the widgets properly. It exists at the top level of the scene graph and can be seen as a root node. All the other nodes (buttons, texts, etc.) need to be added to this layout.
- ☞ In this application, we have implemented **StackPane** layout. It can be implemented by instantiating **javafx.scene.layout.StackPane** class.

```
1.package application;  
2.import javafx.application.Application;  
3.import javafx.scene.control.Button;  
4.import javafx.stage.Stage;  
5.import javafx.scene.layout.StackPane;  
6.public class Hello_World extends Application{  
7.    public void start(Stage primaryStage) throws Exception {  
8.        Button btn1=new Button("Say, Hello World");  
9.        StackPane root=new StackPane();  
10.       root.getChildren().add(btn1);  
11.    }  
12.}
```



Step 4: Create a Scene

- ☞ The layout needs to be added to a scene. Scene remains at the higher level in the hierarchy of application structure. It can be created by instantiating **javafx.scene.Scene** class. We need to pass the layout object to the scene class constructor.
- ☞ We can also pass the width and height of the required stage for the scene in the Scene class constructor.

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Scene;
4.import javafx.scene.control.Button;
5.import javafx.stage.Stage;
6.import javafx.scene.layout.StackPane;
7.public class Hello_World extends Application{
8.    public void start(Stage primaryStage) throws Exception {
9.        Button btn1=new Button("Say, Hello World");
10.        StackPane root=new StackPane();
11.        root.getChildren().add(btn1);
12.        Scene scene=new Scene(root);
13.    }
14.}
```



Step 5: Prepare the Stage

- ☞ **javafx.stage.Stage** class provides some important methods which are required to be called to set some attributes for the stage. We can set the title of the stage. We also need to call `show()` method without which, the stage won't be shown.

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Scene;
4.import javafx.scene.control.Button;
5.import javafx.stage.Stage;
6.import javafx.scene.layout.StackPane;
7.public class Hello_World extends Application{
8.    public void start(Stage primaryStage) throws Exception
on {
9.        Button btn1=new Button("Say, Hello World");
10.        StackPane root=new StackPane();
11.        root.getChildren().add(btn1);
12.        Scene scene=new Scene(root);
13.        primaryStage.setScene(scene);
14.        primaryStage.setTitle("First JavaFX Application");
15.        primaryStage.show();
16.    }
17.}
```



Step 6: Create an event for the button

- ☞ As our application prints hello world for an event on the button. We need to create an event for the button. For this purpose, call **setOnAction()** on the button and define a anonymous class Event Handler as a parameter to the method.

```
1.package application;
2.import javafx.application.Application;
3.import javafx.event.ActionEvent;
4.import javafx.event.EventHandler;
5.import javafx.scene.Scene;
6.import javafx.scene.control.Button;
7.import javafx.stage.Stage;
8.import javafx.scene.layout.StackPane;
9.public class Hello_World extends Application{
10.                                     public
void start(Stage primaryStage) throws Exception {
11.     Button btn1=new Button("Say, Hello World");
12.     btn1.setOnAction(new EventHandler<ActionEvent>()
{
13.         public void handle(ActionEvent arg0) {
14.             System.out.println("hello world");
15.         }
16.     });
17.     StackPane root=new StackPane();
18.     root.getChildren().add(btn1);
19.     Scene scene=new Scene(root,600,400);
20.     primaryStage.setScene(scene);
21.     primaryStage.setTitle("First JavaFX Application");
22.     primaryStage.show();
```




```

1.package application;
2.import javafx.application.Application;
3.import javafx.event.ActionEvent;
4.import javafx.event.EventHandler;
5.import javafx.scene.Scene;
6.import javafx.scene.control.Button;
7.import javafx.stage.Stage;
8.import javafx.scene.layout.StackPane;
9.public class Hello_World extends Application{
10.    public void start(Stage primaryStage) throws Exception {
11.        Button btn1=new Button("Say, Hello World");
12.        btn1.setOnAction(new EventHandler<ActionEvent>
13.            >() {
14.                public void handle(ActionEvent arg0) {
15.                    System.out.println("hello world");
16.                }
17.            });
18.        StackPane root=new StackPane();
19.        root.getChildren().add(btn1);
20.        Scene scene=new Scene(root,600,400);
21.        primaryStage.setTitle("First JavaFX Application");
22.        primaryStage.setScene(scene);
23.        primaryStage.show();
24.    }
25.    public static void main (String[] args)
26.    {
    launch(args);
    }

```

Step 7: Create the main method

- ☞ Till now, we have configured all the necessary things which are required to develop a basic JavaFX application.
- ☞ We have not created main method yet. Hence, at the last, we need to create a main method in which we will launch the application i.e. will call launch() method and pass the command line arguments (args) to it.



- ➡ The Application will produce this output on the screen.



My JavaFx

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

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

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

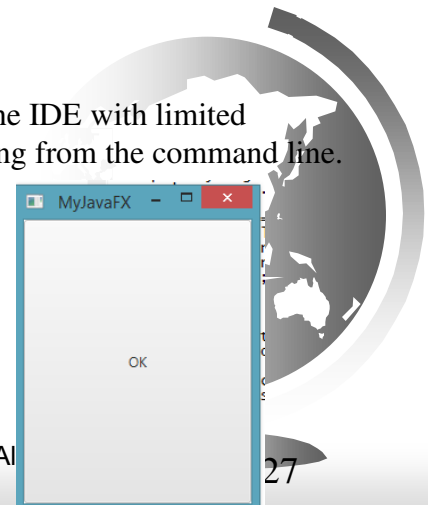
MultipleStage Demo

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

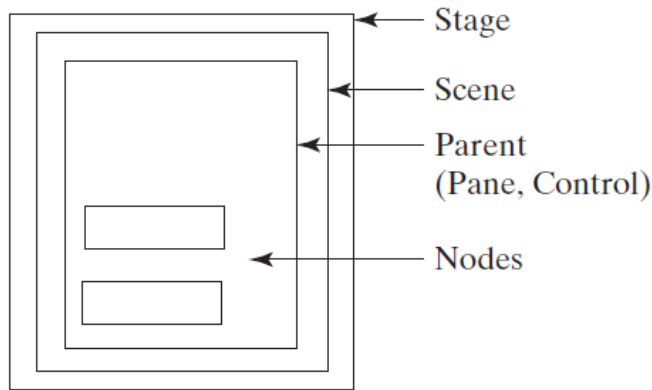
public class MultipleStageDemo extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Scene scene = new Scene(new Button("OK"), 200, 250);
        primaryStage.setTitle("MyJavaFX");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

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

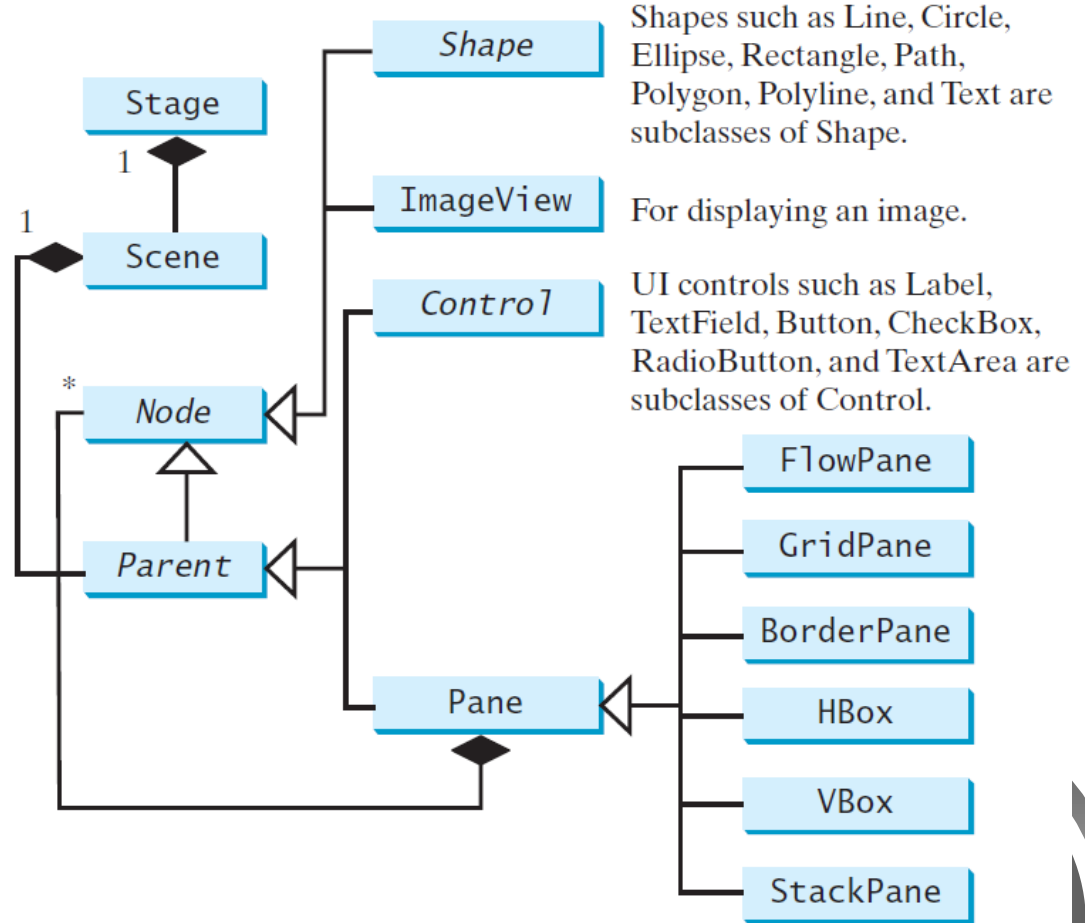
    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    //public static void main(String[] args) {
    //    launch(args);
    //}
}
```



Panes, UI Controls, and Shapes



(a)



(b)

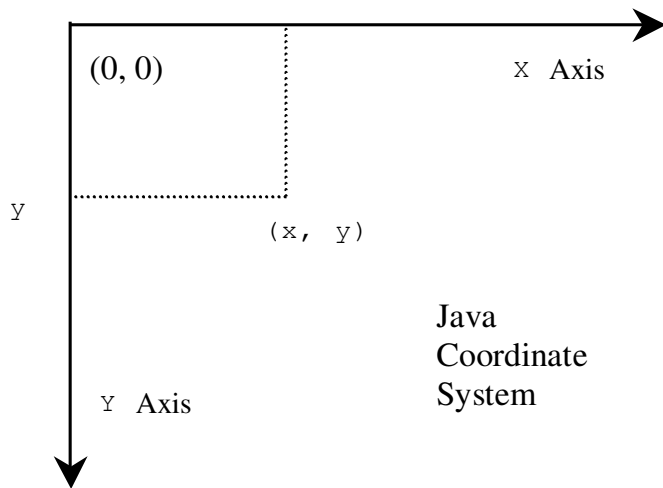


ButtonInPane

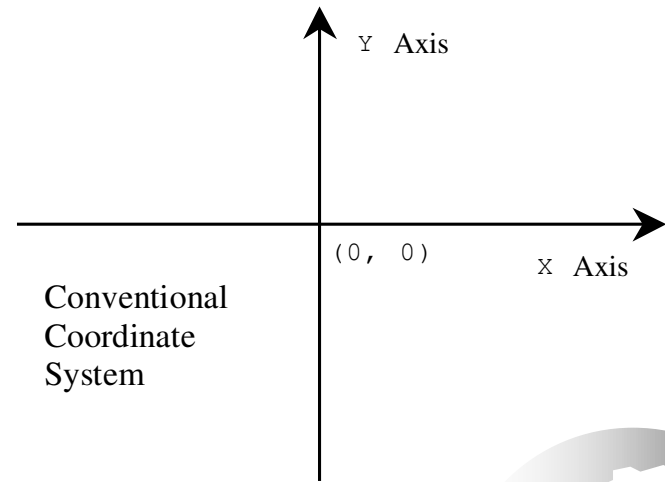
Run

Display a Shape

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



Java
Coordinate
System



Conventional
Coordinate
System



ShowCircle

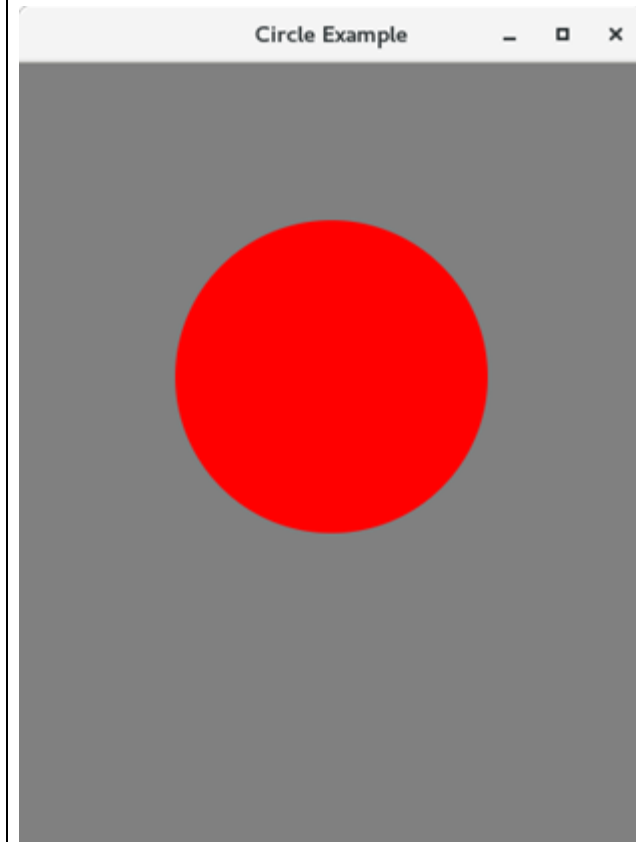
Run

➡ JavaFX Circle

- ➡ JavaFX allows us to create Circle on the GUI of any application by just instantiating **`javafx.scene.shape.Circle`** class.

Property	Description	Setter Methods
centerX	X coordinate of the centre of circle	setCenterX(Double value)
centerY	Y coordinate of the centre of circle	setCenterY(Double value)
radius	Radius of the circle	setRadius(Double value)

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Group;
4.import javafx.scene.Scene;
5.import javafx.scene.paint.Color;
6.import javafx.scene.shape.Circle;
7.import javafx.stage.Stage;
8.public class Shape_Example extends Application{
9.    public void start(Stage primaryStage) throws Excepti
on {
10.        primaryStage.setTitle("Circle Example");
11.        Group group = new Group();
12.        Circle circle = new Circle();
13.        circle.setCenterX(200);
14.        circle.setCenterY(200);
15.        circle.setRadius(100);
16.        circle.setFill(Color.RED);
17.        group.getChildren().addAll(circle);
18.        Scene scene = new Scene(group,400,500,Color.GR
AY);
19.        primaryStage.setScene(scene);
20.        primaryStage.show();
21.}
22.public static void main(String[] args) {
23.    launch(args);
24.}
25.
```



Binding Properties


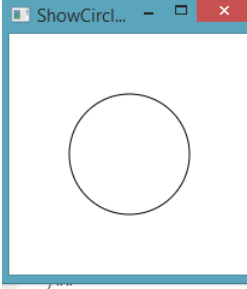
JavaFX introduces a new concept called *binding property* that enables a *target object* to be bound to a *source object*. If the value in the source object changes, the target property is also changed automatically. The target object is simply called a *binding object* or a *binding property*.

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

public class ShowCircleCentered extends Application {
    public void start(Stage primaryStage) {
        // Create a pane to hold the circle
        Pane pane = new Pane();
        // Create a circle and set its properties
        Circle circle = new Circle();
        circle.centerXProperty().bind(pane.widthProperty().divide(2));
        circle.centerYProperty().bind(pane.heightProperty().divide(2));
        circle.setRadius(50);
        circle.setStroke(Color.BLACK);
        circle.setFill(Color.WHITE);
        pane.getChildren().add(circle); // Add circle to the pane

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 200, 200);
        primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

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



Binding Property: getter, setter, and property getter

```
public class SomeClassName {  
  
    private PropertyType x;  
  
    /** Value getter method */  
    public PropertyValue getX() { ... }  
  
    /** Value setter method */  
    public void setX(PropertyType value) { ... }  
  
    /** Property getter method */  
    public PropertyType  
        xProperty() { ... }  
}
```

(a) x is a binding property

```
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



The Color Class

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

javafx.scene.paint.Color

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

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

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

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

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

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

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

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

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

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

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

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

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

RGB Color

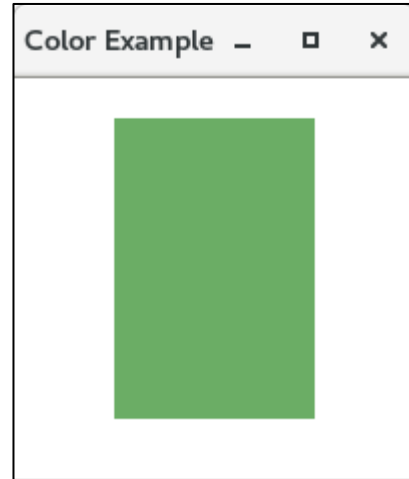
RGB color system is the most popular method to create a color in graphics. It consists of three components named as RED → R, GREEN → G and BLUE → B. Each component uses 8 Bits that means every component can have the integer value from 0 to $2^8 - 1 = 255$.

There is a static method named as **rgb()** of Color class. It accepts three integer arguments as Red, Green, Blue and one optional double argument called

```
4.import javafx.scene.Scene;
5.import javafx.scene.effect.DropShadow;
6.import javafx.scene.effect.Shadow;
7.import javafx.scene.paint.Color;
8.import javafx.scene.shape.Rectangle;
9.import javafx.stage.Stage;
10.public class Shape_Example extends Appli
cation
```

```
11. public void start(Stage primarystage) {
12.    Group root = new Group();
13.    primarystage.setTitle("Color Example");
14.    Rectangle rect = new Rectangle();
15.    rect.setX(50);
16.    rect.setY(20);
17.    rect.setWidth(100);
18.    rect.setHeight(150);
19.    int red=20;
20.    int green=125;
21.    int blue=10;
22.    rect.setFill(Color.rgb(red, green, blue,0.63)
);
23.    root.getChildren().add(rect);
24.    Scene scene = new Scene(root,200,200);

25.    primarystage.setScene(scene);
26.    primarystage.show();
27. }
28. public static void main(String[] args) {
29.    launch(args);
}
```



The Font Class

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

javafx.scene.text.Font

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

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

The size of this font.

The name of this font.

The family of this font.

Creates a Font with the specified size.

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

Creates a Font with the specified name and size.

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

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

Returns a list of font family names.

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



FontDemo

Run

The Image Class

javafx.scene.image.Image

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

+Image(filenameOrURL: String)

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

Indicates whether the image is loaded correctly?

The height of the image.

The width of the image.

The approximate percentage of image's loading that is completed.

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



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

Layout Panes

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

<i>Class</i>	<i>Description</i>
Pane	Base class for layout panes. It contains the 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

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

javafx.scene.layout.FlowPane

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

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

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

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

Creates a default FlowPane.

Creates a FlowPane with a specified horizontal and vertical gap.

Creates a FlowPane with a specified orientation.

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



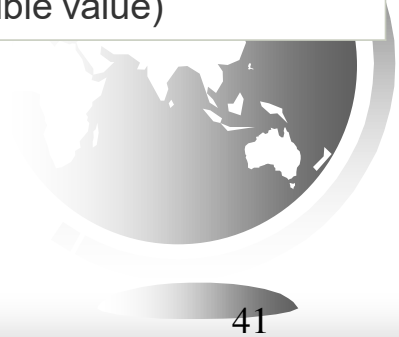
ShowFlowPane

Run

JavaFX FlowPane

- FlowPane layout pane organizes the nodes in a flow that are wrapped at the flowpane's boundary. The horizontal flowpane arranges the nodes in a row and wrap them according to the flowpane's width. The vertical flowpane arranges the nodes in a column and wrap them according to the flowpane's height.

Property	Description	Setter Methods
alignment	The overall alignment of the flowpane's content.	setAlignment(Pos value)
columnHalignment	The horizontal alignment of nodes within the columns.	setColumnHalignment(HPos Value)
hgap	Horizontal gap between the columns.	setHgap(Double value)
orientation	Orientation of the flowpane	setOrientation(Orientation value)
prefWrapLength	The preferred height or width where content should wrap in the horizontal or vertical flowpane.	setPrefWrapLength(double value)
rowValignment	The vertical alignment of the nodes within the rows.	setRowValignment(VPos value)
vgap	The vertical gap among the rows	setVgap(Double value)



FlowPane

Constructors

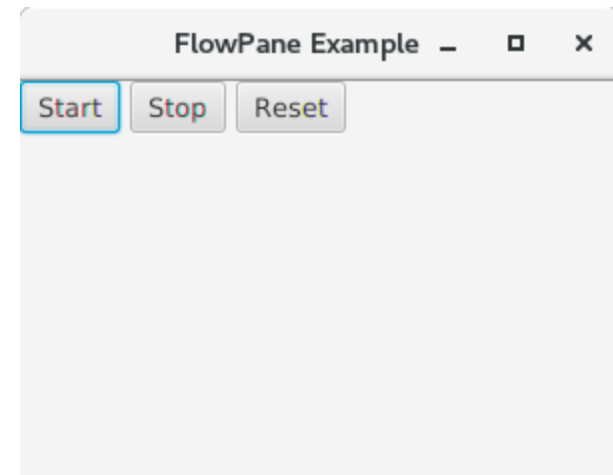
There are 8 constructors in the class that are given below.

1. `FlowPane()`
2. `FlowPane(Double Hgap, Double Vgap)`
3. `FlowPane(Double Hgap, Double Vgap, Node? children)`
4. `FlowPane(Node... Children)`
5. `FlowPane(Orientation orientation)`
6. `FlowPane(Orientation orientation, double Hgap, Double Vgap)`
7. `FlowPane(Orientation orientation, double Hgap, Double Vgap, Node? children)`
8. `FlowPane(Orientation orientation, Node... Children)`



FlowPane

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Scene;
4.import javafx.scene.control.Button;
5.import javafx.scene.layout.FlowPane;
6.import javafx.stage.Stage;
7.public class FlowPaneTest extends Application
{
8.    public void start(Stage primaryStage) {
9.        primaryStage.setTitle("FlowPane Example");
10.        FlowPane root = new FlowPane();
11.        root.setVgap(6);
12.        root.setHgap(5);
13.        root.setPrefWrapLength(250);
14.        root.getChildren().add(new Button("Start"));
15.        root.getChildren().add(new Button("Stop"));
16.        root.getChildren().add(new Button("Reset"));
17.        Scene scene = new Scene(root,300,200);
18.        primaryStage.setScene(scene);
19.        primaryStage.show();
20.    }
21.    public static void main(String[] args) {
```



GridPane

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

javafx.scene.layout.GridPane

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

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

The overall alignment of the content in this pane (default: Pos.LEFT).
Is the grid line visible? (default: false)

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

Creates a GridPane.

Adds a node to the specified column and row.

Adds multiple nodes to the specified column.

Adds multiple nodes to the specified row.

Returns the column index for the specified node.

Sets a node to a new column. This method repositions the node.

Returns the row index for the specified node.

Sets a node to a new row. This method repositions the node.

Sets the horizontal alignment for the child in the cell.

Sets the vertical alignment for the child in the cell.

ShowGridPane

Run

JavaFX GridPane

GridPane Layout pane allows us to add the multiple nodes in multiple rows and columns. It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid.

Property	Description	Setter Methods
alignment	Represents the alignment of the grid within the GridPane.	setAlignment(Pos value)
gridLinesVisible	This property is intended for debugging. Lines can be displayed to show the gridpane's rows and columns by setting this property to true.	setGridLinesVisible(Boolean value)
hgap	Horizontal gaps among the columns	setHgap(Double value)
vgap	Vertical gaps among the rows	setVgap(Double value)

Constructors

The class contains only one constructor:

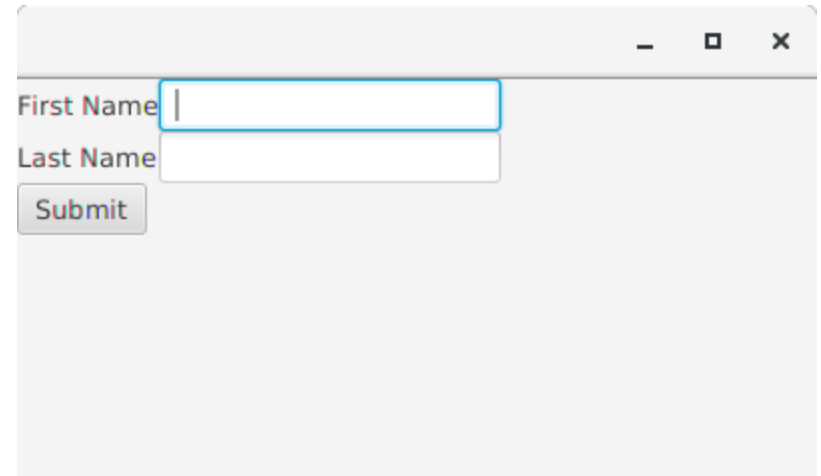
Public GridPane(): creates a gridpane with 0 hgap/vgap.



GridPane

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Scene;
4.import javafx.scene.control.Button;
5.import javafx.scene.control.Label;
6.import javafx.scene.control.TextField;
7.import javafx.scene.layout.GridPane;
8.import javafx.stage.Stage;
9.public class Label_Test extends Application {

10.    public void start(Stage primaryStage) throws Exception {
11.        Label first_name=new Label("First Name");
12.        Label last_name=new Label("Last Name");
13.        TextField tf1=new TextField();
14.        TextField tf2=new TextField();
15.        Button Submit=new Button ("Submit");
16.        GridPane root=new GridPane();
17.        Scene scene = new Scene(root,400,200);
18.        root.addRow(0, first_name,tf1);
19.        root.addRow(1, last_name,tf2);
20.        root.addRow(2, Submit);
21.        primaryStage.setScene(scene);
22.        primaryStage.show();
23.    }
```



BorderPane

`javafx.scene.layout.BorderPane`

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

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

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

The node placed in the top region (default: `null`).
The node placed in the right region (default: `null`).
The node placed in the bottom region (default: `null`).
The node placed in the left region (default: `null`).
The node placed in the center region (default: `null`).

Creates a `BorderPane`.

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



ShowBorderPane

Run

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.



HBox

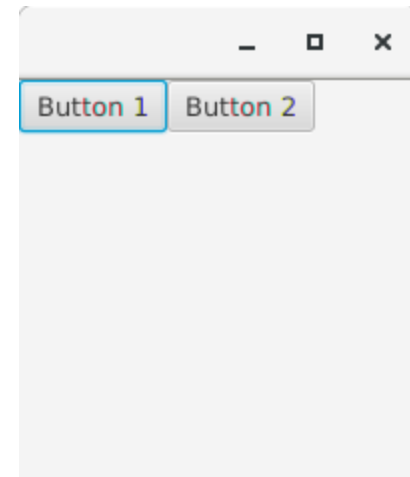
JavaFX HBox

HBox layout pane arranges the nodes in a single row. It is represented by **`javafx.scene.layout.HBox`** class

Property	Description	Setter Methods
alignment	This represents the alignment of the nodes.	setAlignment(Double)
fillHeight	This is a boolean property. If you set this property to true the height of the nodes will become equal to the height of the HBox.	setFillHeight(Double)
spacing	This represents the space between the nodes in the HBox. It is of double type.	setSpacing(Double)

HBox

```
1.package application;
2.import javafx.application.Application;
3.import javafx.scene.Scene;
4.import javafx.scene.control.Button;
5.import javafx.scene.layout.HBox;
6.import javafx.stage.Stage;
7.public class Label_Test extends Application {
8.public void start(Stage primaryStage) throws Exception {
9.Button btn1 = new Button("Button 1");
10.Button btn2 = new Button("Button 2");
11.HBox root = new HBox();
12.Scene scene = new Scene(root,200,200);
13.root.getChildren().addAll(btn1,btn2);
14.primaryStage.setScene(scene);
15.primaryStage.show();
16.}
17.public static void main(String[] args) {
18.    launch(args);
19.}
```



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.

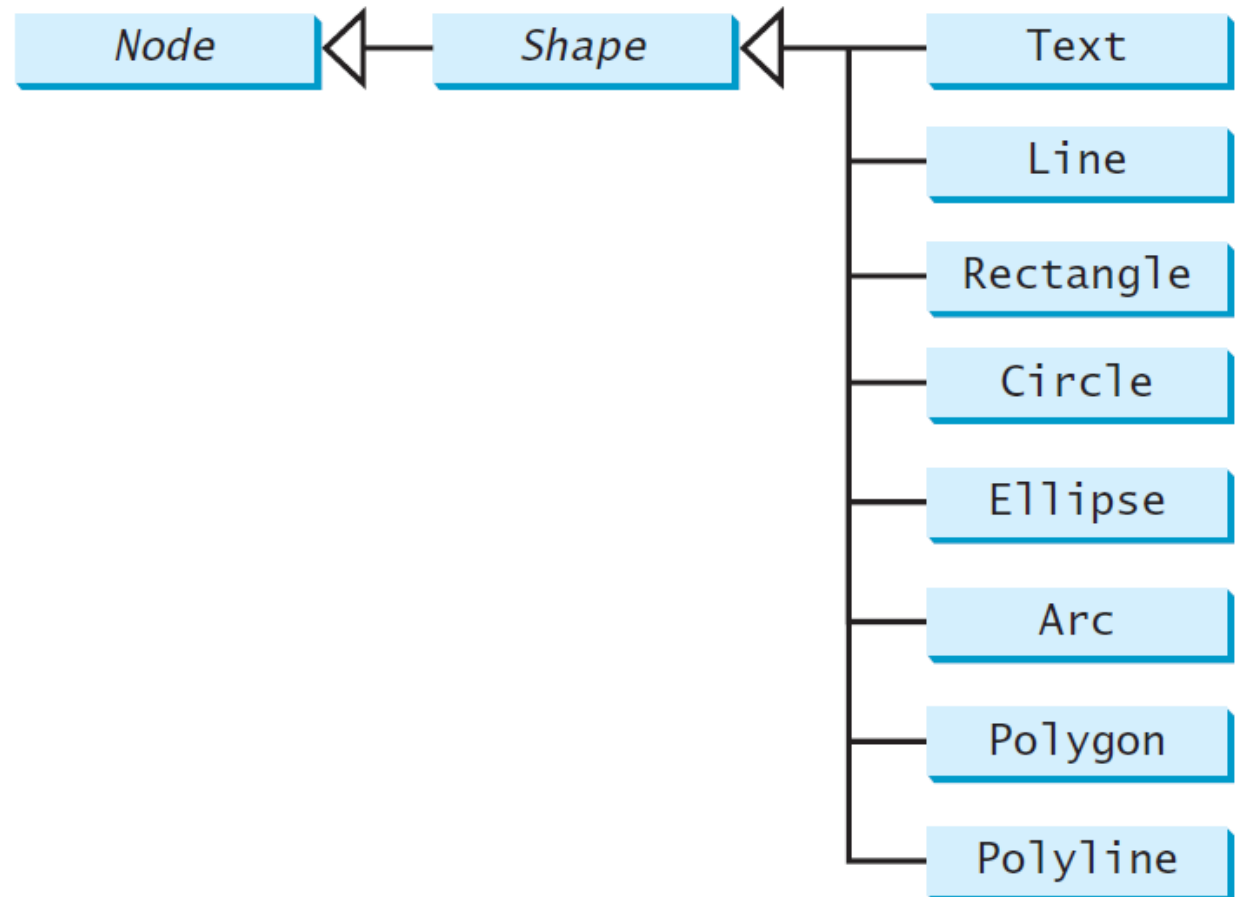


ShowHBoxVBox

Run

Shapes

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



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

+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).

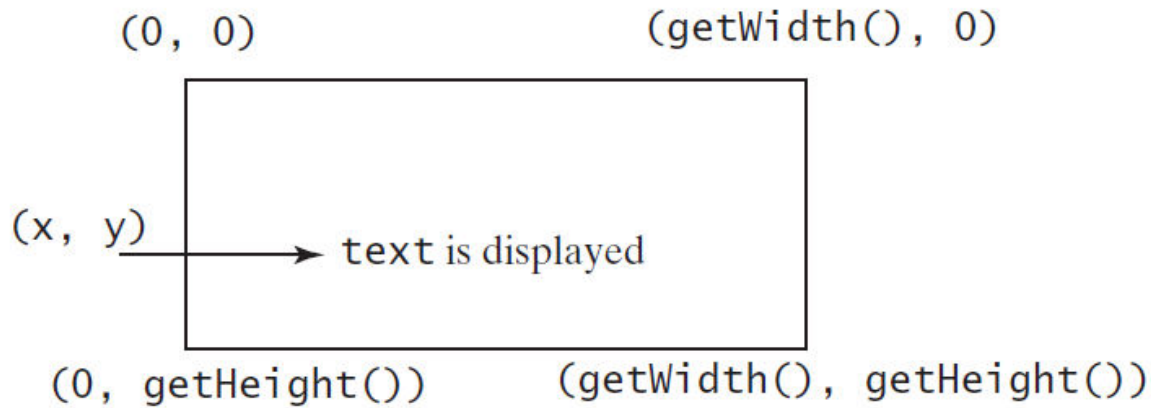
Defines the font for the text.

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

Run

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.

(0, 0)

(getWidth(), 0)

(startX, startY)

(endX, endY)

(0, getHeight())

(getWidth(), getHeight())

ShowLine

Run

Rectangle

javafx.scene.shape.Rectangle

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

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

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

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

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

The width of the rectangle (default: 0).

The height of the rectangle (default: 0).

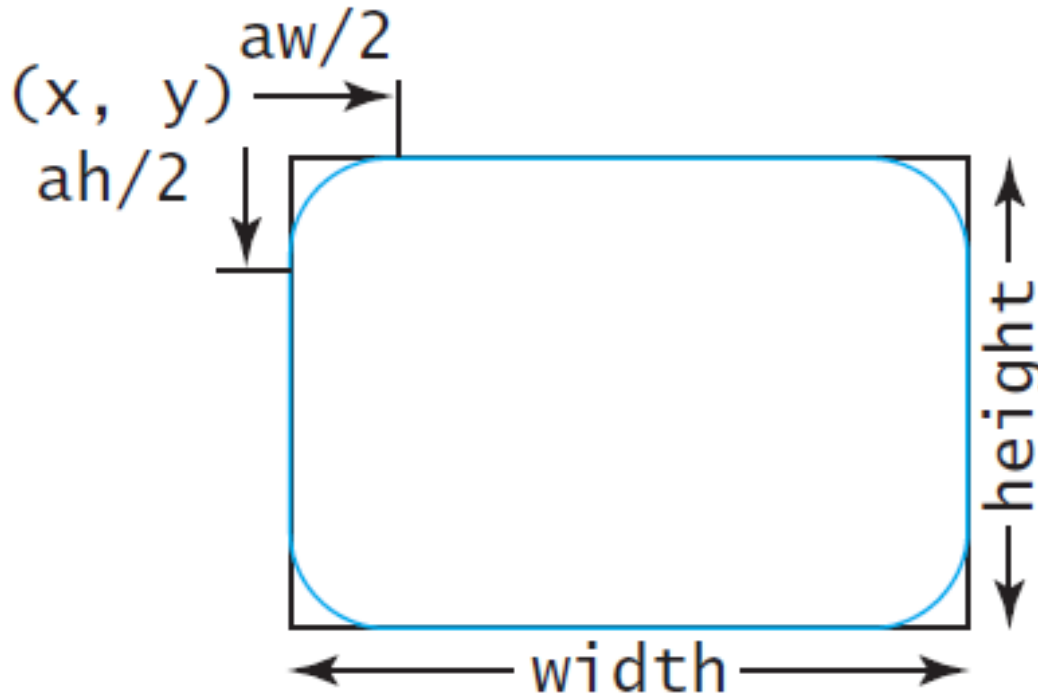
The arcWidth of the rectangle (default: 0). arcWidth is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

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

Creates an empty Rectangle.

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

Rectangle Example



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



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

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

javafx.scene.shape.Ellipse

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

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

The x-coordinate of the center of the ellipse (default 0).

The y-coordinate of the center of the ellipse (default 0).

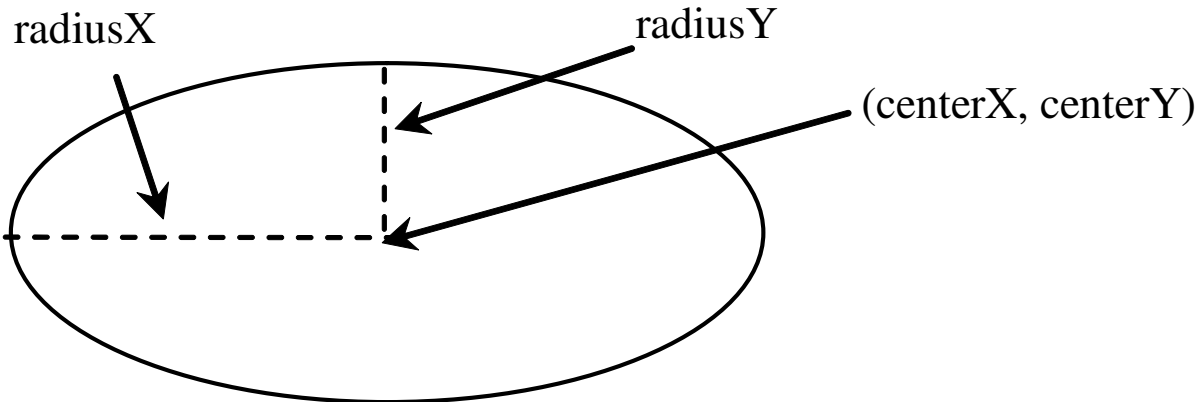
The horizontal radius of the ellipse (default: 0).

The vertical radius of the ellipse (default: 0).

Creates an empty `Ellipse`.

Creates an `Ellipse` with the specified center.

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



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.

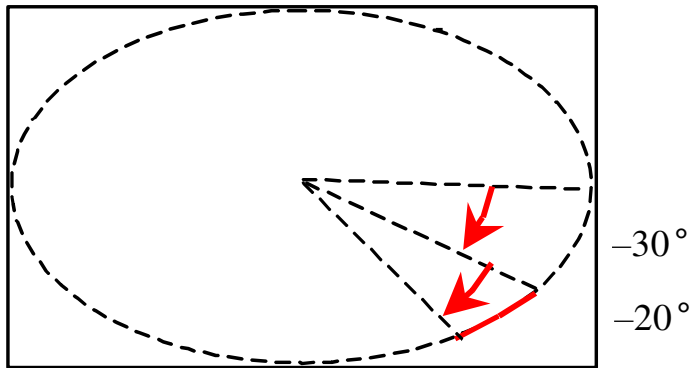
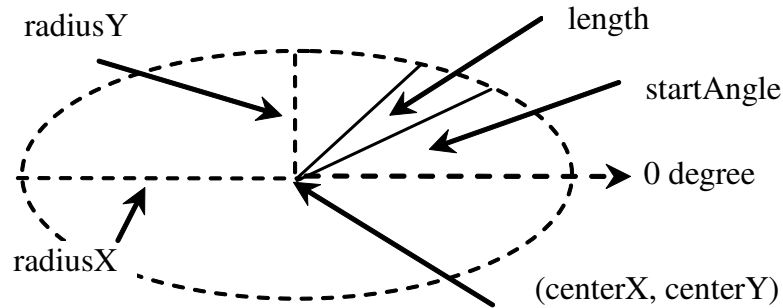
The angular extent of the arc in degrees.

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

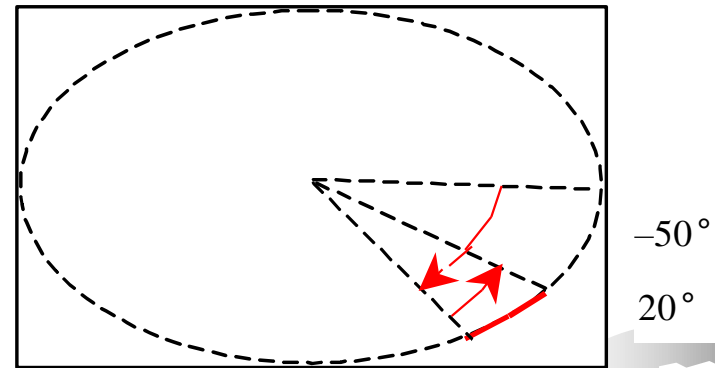
Creates an empty Arc.

Creates an Arc with the specified arguments.

Arc Examples



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



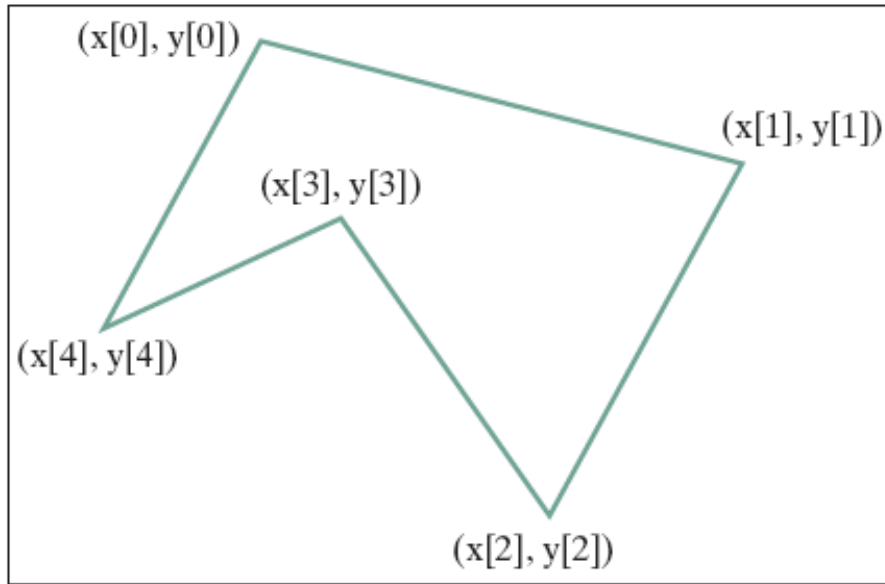
(b) Negative starting angle -50° and positive spanning angle 20°



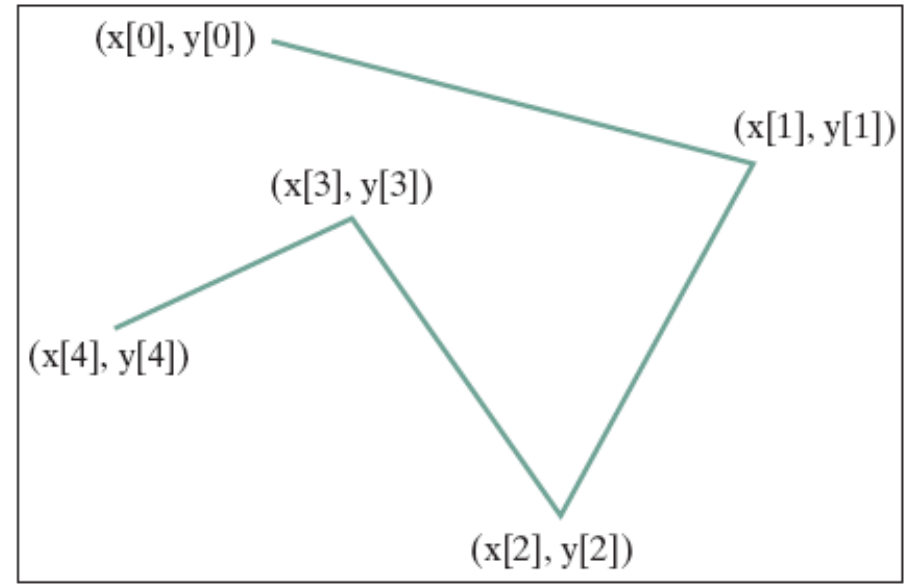
ShowArc

Run

Polygon and Polyline



(a) Polygon



(b) Polyline



ShowArc

Run

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.

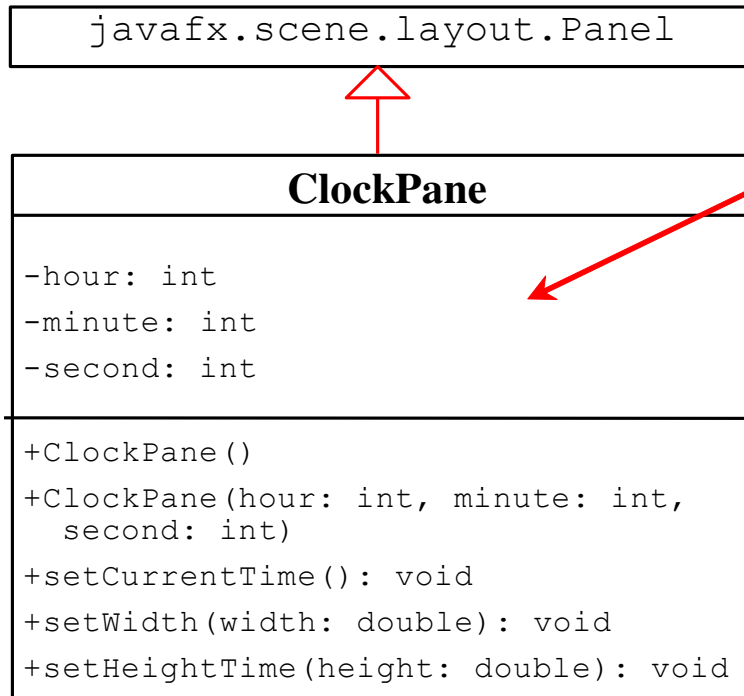


ShowPolygon

Run

Case Study: The ClockPane Class

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



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

The hour in the clock.

The minute in the clock.

The second in the clock.

Constructs a default clock for the current time.

Constructs a clock with the specified time.

Sets hour, minute, and second for current time.

Sets clock pane's width and repaint the clock,

Sets clock pane's height and repaint the clock,

