14013103-4

ADVANCED PROGRAMMING

LECTURE 2

GUI using JAVAFX

Created by Randah Alharbi & Bushra Algotiml

Outline

- Graphical User Interface
    - How do GUIs works
    - Java APIs for graphics programming
    - Differences between JavaFX, Swing, and AWT
- JavaFX Overview
- JavaFX Application
    - Application Structure: **Stage, Scene, Scene Graph, Nodes**
    - Creating JavaFX Application
    - Lifecycle and terminating of JavaFX Application
    - Example

Outline

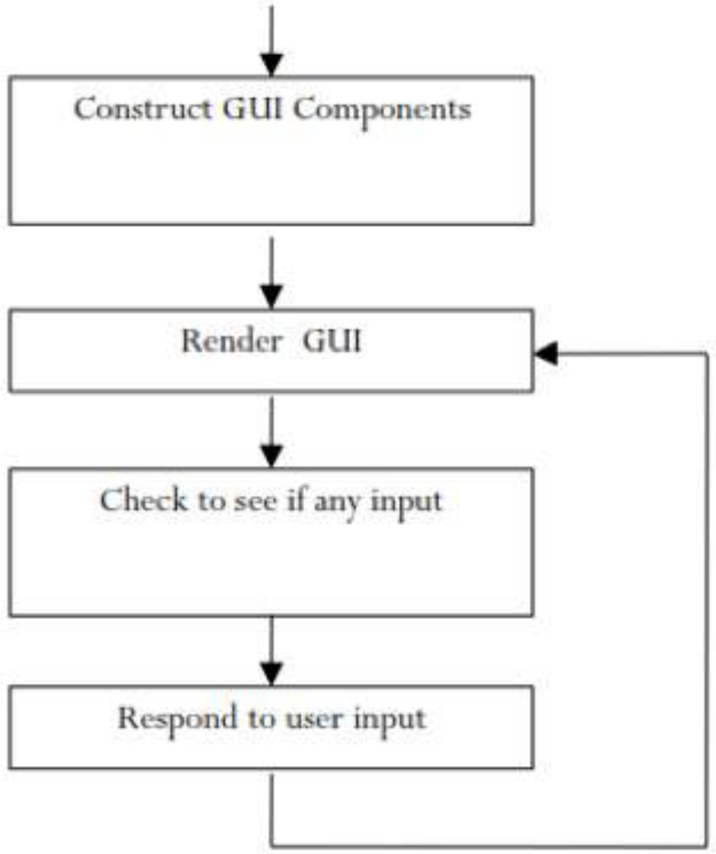
- JavaFX Nodes:
  - **Shapes**: Text, Line, Circle, Rectangle
  - **Font**
  - **Colors**
  - **Images**
  - **Layout Panes**: Pane, StackPane, FlowPane, GridPane, BorderPane, HBox, and VBox
  - **UI Controls**: Label, Button, CheckBox, RadioButton, TextField, TextArea, ComboBox, ListView, ScrollBar, and Slider.
- Extra Reading: Effects, Transformation, Animation, JavaFX CSS and MediaPlayer (Audio and Video).

Graphical User Interface and how it works

- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an app. A GUI (pronounced “GOO-ee”) gives an app a distinctive “look-and-feel.” GUIs are built from GUI components—also called controls or widgets (short for window gadgets). A GUI component is an object with which the user interacts via the mouse, the keyboard or another form of input, such as voice recognition.
- GUIs loop and respond to events
- Example: a mouse click on a button

  A. Operating System recognizes mouse click
    - Determines which window it was inside
    - Notifies that program

  B. Program runs in loop
  - Checks input buffer filled by OS
  - If it finds a mouse click:
    - Determines which component in the program
    - If the click was on a relevant component
      - respond appropriately according to handler



Java APIs for graphics programming

- JDK (Java Development Kit) provides pre-built graphics classes for constructing your own Graphical User Interface (GUI) applications.
- These graphics classes, developed by expert programmers, are highly complex and involve many advanced design patterns. However, re-using them are not so difficult, if you follow the API documentation, samples and templates provided.
- Currently, there are three sets of Java APIs for graphics programming: **AWT**(Abstract Windowing Toolkit), **Swing** and **JavaFX**.
  1. **AWT** API was introduced in JDK 1.0. Most of the AWT components have become old and should be replaced by newer Swing components.
  2. **Swing** API, a much more comprehensive set of graphics libraries that enhances the AWT.
  3. The latest **JavaFX**, which was integrated into **JDK 8**, is meant to replace Swing.

Differences between JavaFX, Swing, and AWT

- AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
- JavaFX is easier to use—it provides one API for GUI, graphics and multimedia (images, animation, audio and video), whereas Swing is only for GUIs, so you need to use other APIs for graphics and multimedia apps.
- With Swing, many IDEs provided GUI design tools for dragging and dropping components onto a layout; however, each IDE produced different code. JavaFX Scene Builder can be used standalone or integrated with many IDEs and it produces the same code regardless of the IDE.
- Though Swing components could be customized, JavaFX gives you complete control over a JavaFX GUI’s look-and-feel via Cascading Style Sheets (CSS).

JavaFX Overview

- JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

Features of JavaFX:

- **Written in Java:** The JavaFX library is written in Java and is available for the languages that can be executed on a JVM, which include – Java, Groovy and JRuby. These JavaFX applications are also platform independent.
- **FXML:** JavaFX features a language known as FXML, which is a HTML like declarative markup language. The sole purpose of this language is to define a user Interface.
- **Scene Builder:** JavaFX provides an application named Scene Builder. On integrating this application in IDE's such as Eclipse and NetBeans, the users can access a drag and drop design interface, which is used to develop FXML applications (just like Swing Drag & Drop and DreamWeaver Applications).

JavaFX Overview

- **Built-in UI controls:** JavaFX library caters UI controls using which we can develop a full-featured application.
- **CSS like Styling:** JavaFX provides a CSS like styling. By using this, you can improve the design of your application with a simple knowledge of CSS.
- **Rich set of API's:** JavaFX provides a complete API with a rich set of classes and interfaces to build GUI applications with rich graphics such as: 2D and 3D graphics, Media, UI controls, Web, Graphics Animation, Effects, Event handling, etc.

JAVAFX Application

Stage, Scene, Scene Graph, Nodes

JavaFX Application

- Application Structure
  - Stage
  - Scene
  - Scene Graph and Nodes
- Creating JavaFX Application
  - Application class
  - Preparing the Scene Graph
  - Preparing the Scene
  - Preparing the stage
- Lifecycle of JavaFX Application
- Terminating the JavaFX Application

Example – Creating an Empty Window

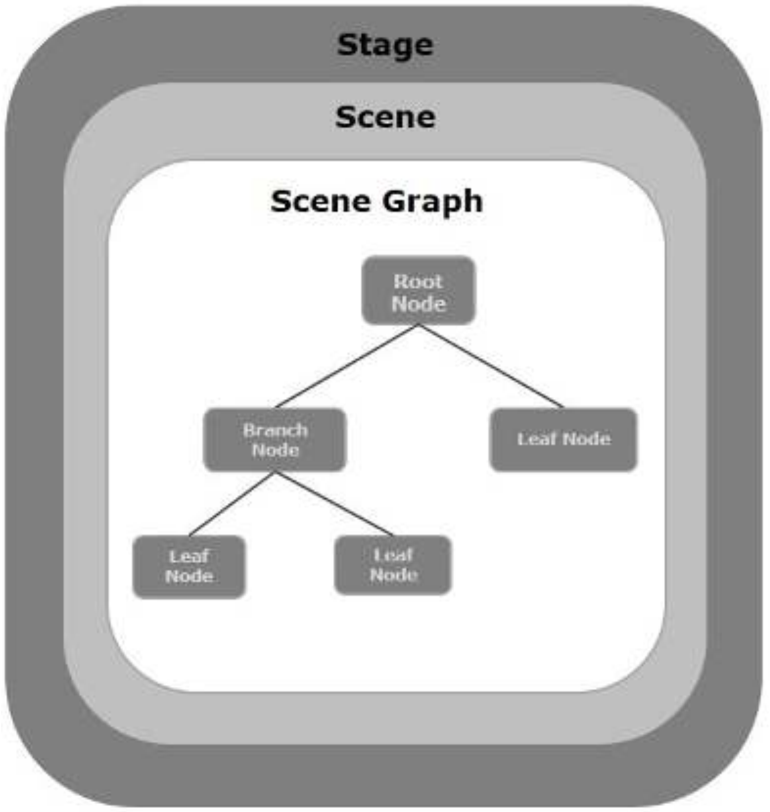
```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavafxSample extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {

        Group group = new Group(); //creating a Group object
        Scene scene = new Scene(group ,600, 300); //Creating a Scene by passing the group object,
                                                    height and width
        scene.setFill(Color.RED); //setting color to the scene
        primaryStage.setTitle("Sample Application"); //Setting the title to Stage.
        primaryStage.setScene(scene); //Adding the scene to Stage
        primaryStage.show(); //Displaying the contents of the stage
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

JavaFX Application - Stage

- In general, a JavaFX application will have three major components namely **Stage, Scene** and **Nodes** as shown in the next diagram.

Stage:

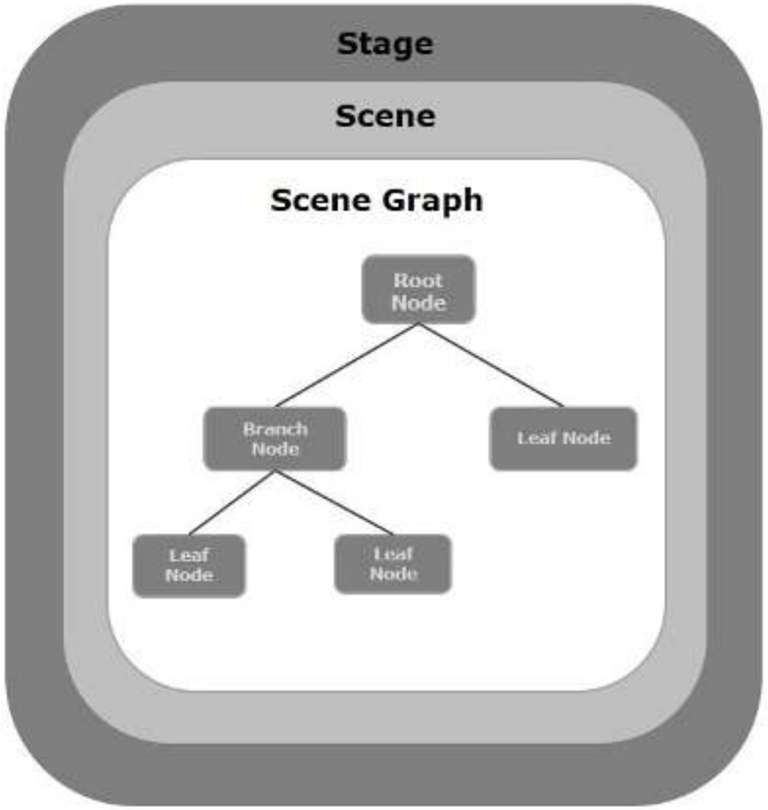
- A **stage** (a window) contains all the objects of a JavaFX application.
- It is represented by **Stage** class of the package `javafx.stage`.
- The primary stage is created by the platform itself. The created stage object is passed as an argument to the `start()` method of the `Application` class.



JavaFX Application - Stage

Stage:

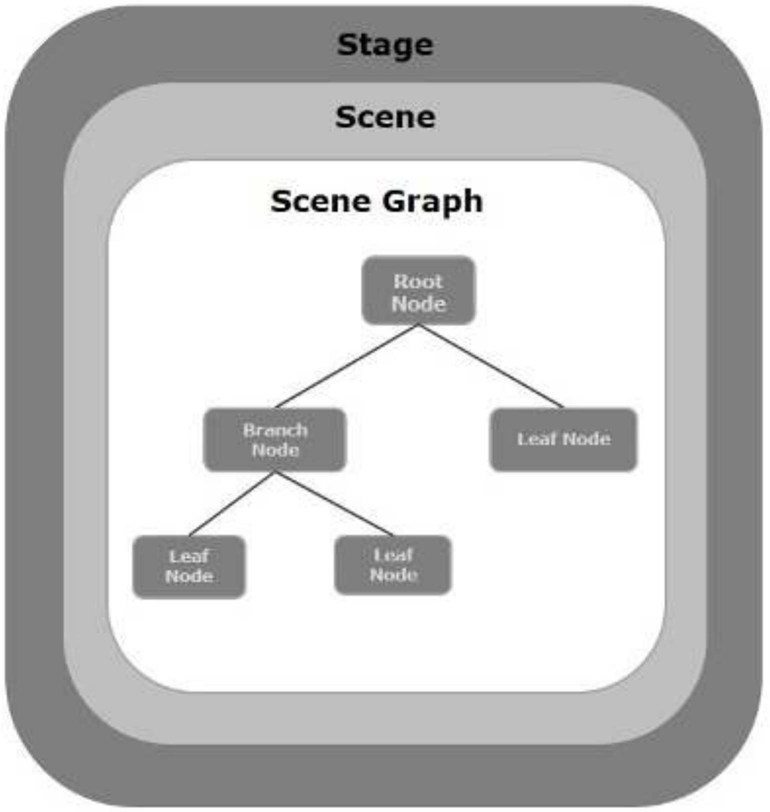
- A **stage** has two parameters determining its position namely **Width** and **Height**. It is divided as Content Area and Decorations (Title Bar and Borders).
- There are five types of stages available (Decorated, Undecorated, Transparent, Unified, Utility)
- You have to call the `show()` method to display the contents of a stage.



JavaFX Application - Scene

Scene:

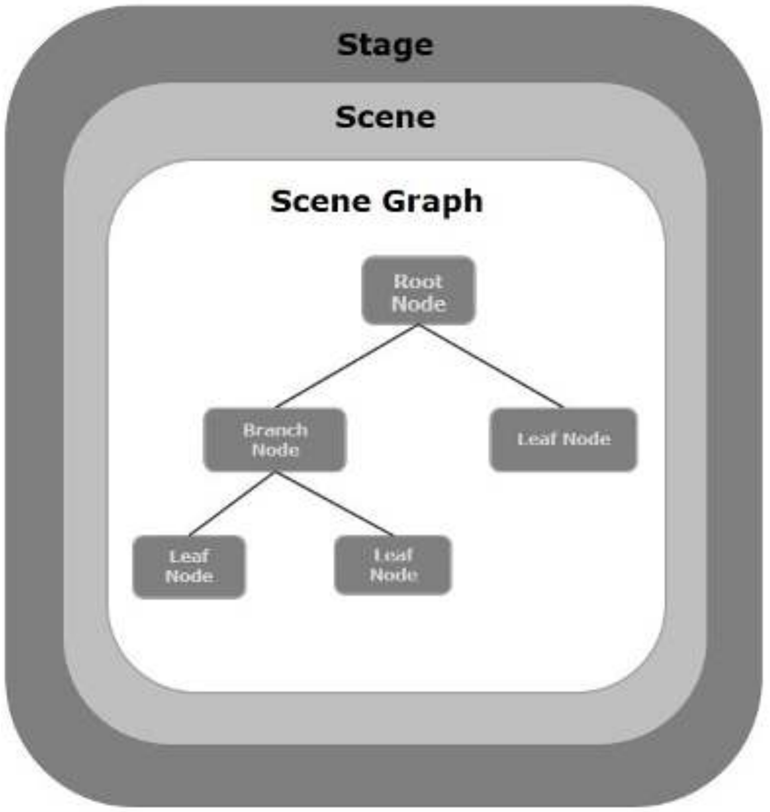
- A **scene** represents the physical contents of a JavaFX application.
- It contains all the contents of a scene graph.
- The class **Scene** of the package `javafx.scene` represents the scene object. At an instance, the scene object is added (passed) to only one stage.
- You can create a scene by instantiating the `Scene` Class.
- You can opt for the size of the scene by passing its dimensions (**height** and **width**) along with the **root node** to its constructor.



JavaFX Application – Scene Graph and Node

Scene Graph:

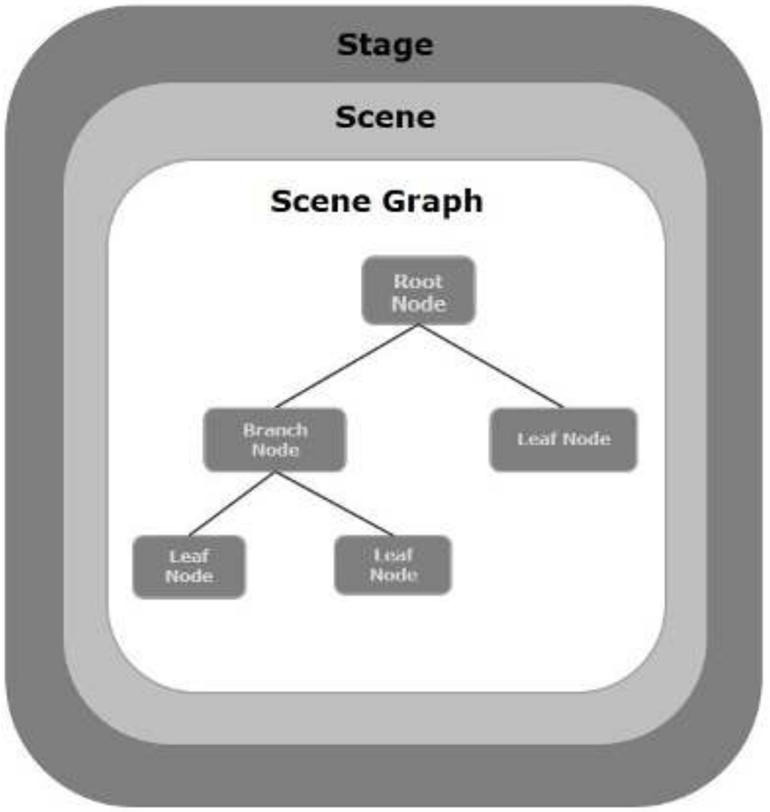
- In JavaFX, the GUI Applications were coded using a Scene Graph. A Scene Graph is the starting point of the construction of the GUI Application. It holds the (GUI) application primitives that are termed as **nodes**.
- Scene Graph API makes GUI easier to create, especially when complex visual effects and transformations are involved.
- A **scene graph** is a tree-like data structure (hierarchical) representing the contents of a scene. In contrast, a **node** is a visual/graphical object of a scene graph.



JavaFX Application – Scene Graph and Node

Node:

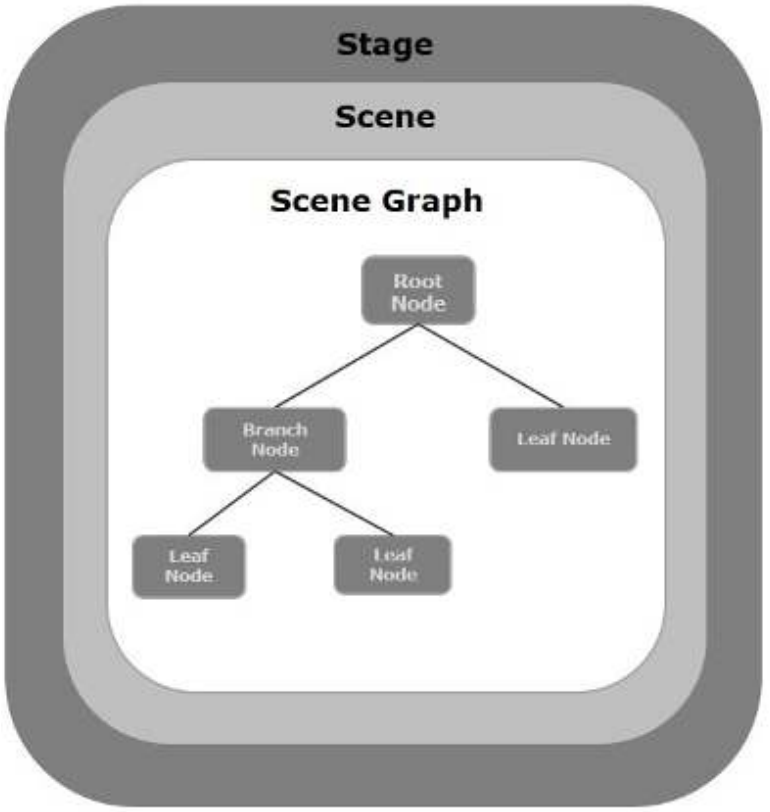
- The individual items held within the JavaFX scene graph are known as **nodes**, it may include:
  - **Geometrical (Graphical) objects –** (2D and 3D) such as circle, rectangle, polygon, etc.
  - **UI controls –** such as Button, Checkbox, Choice box, Text Area, etc.
  - **Containers** – (layout panes) such as Border Pane, Grid Pane, Flow Pane, etc.
  - **Media elements** – such as audio, video and image objects.
- In general, **a collection of nodes makes a scene graph**. All these nodes are arranged in a hierarchical order.
- A node instance can be added to a scene graph only once.



JavaFX Application – Scene Graph and Node

Node:

- The nodes of a scene graph can have Effects, Opacity, Transforms, Event Handlers, Application Specific States.
- The `Node` Class of the package `javafx.scene` represents a node in JavaFX, this class is the super class of all the nodes.
- Each node is one of three types:
  - A **root node** (the first node in the tree and it never has a parent).
  - A **branch node/Parent Node** (meaning that it can have children).
  - A **leaf node** (meaning that it cannot have children).



JavaFX Application – Scene Graph and Node

- **Root Node:** The first node in the Scene Graph is known as the Root node.
- **Branch Node/Parent Node:** The node with child nodes are known as branch/parent nodes.
- The abstract class named **Parent** of the package `javafx.scene` is the base class of all the parent nodes, and those parent nodes will be of the following types:
  - **Group:** A group node is a collective node that contains a list of children nodes.
  - **Panes:** Are container classes, called panes, for automatically laying out the nodes in a desired location and size. You place nodes inside a pane and then place the pane into a scene.
- **Leaf Node:** The node without child nodes is known as the leaf node. For example, Rectangle, Ellipse, Text, ImageView, MediaView are examples of leaf nodes.
- It is mandatory to pass the **root node** of the **scene graph** to the **scene**. If the Group is passed as root, all the nodes will be clipped to the scene and any alteration in the size of the scene will not affect the layout of the scene.
- The root node can be any type of the parent nodes. Usually Group or Pane.

Creating a JavaFX Application

- The `Application` class of the package `javafx.application` is the entry point of the application in JavaFX.
- To create a JavaFX application, you need to **inhert (extend)** the `Application` class and **implement** its abstract method `start()`.
- In `start()` method, you need to write the entire code for the JavaFX graphics.
- In the `main` method, you have to launch the application using the `launch()` method. This method internally calls the `start()` method of the `Application` class.

```
public class JavafxSample extends Application {
    @Override
        public void start(Stage primaryStage) throws Exception {
        /* Code for JavaFX application. (Stage, scene, scene graph) */
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

Creating a JavaFX Application

- Within the `start()` method, in order to create a typical JavaFX application, you need to follow the steps given below:

  1- Prepare a **scene graph** with the required **nodes**.

  2- Prepare a **Scene** with the required dimensions and add the scene graph to it.

  3- Prepare a **stage** and add the scene to the stage and display the contents of the stage.

Example – Creating an Empty Window

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavafxSample extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {

        Group group = new Group(); //creating a Group object
        Scene scene = new Scene(group ,600, 300); //Creating a Scene by passing the group object,
                                                     height and width
        scene.setFill(Color.RED); //setting color to the scene
        primaryStage.setTitle("Sample Application"); //Setting the title to Stage.
        primaryStage.setScene(scene); //Adding the scene to Stage
        primaryStage.show(); //Displaying the contents of the stage
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

Creating a JavaFX Application
1- Preparing the Scene Graph

- As per your application, you need to prepare a scene graph with required nodes.
- Since the root node is the first node, you need to create a root node.
- As a root node, you can choose either **Group** or any kind of **Pane**.

- Group:

```
Group root = new Group();
```

- Pane:

```
StackPane root_pane = new StackPane();
```

Creating a JavaFX Application
2- Preparing the Scene

- A JavaFX scene is represented by the **Scene** class of the package `javafx.scene`. You can create a Scene by instantiating this class.
- While instantiating, it is **mandatory** to pass the root object (the root of the scene graph) to the constructor of the scene class.

```
Scene scene = new Scene(root);
```

- You can also pass two parameters of double type representing the height and width of the scene.

```
Scene scene = new Scene(root, 600, 300);
```

Creating a JavaFX Application
3- Preparing the Stage

- Stage is represented by **Stage** class of the package `javafx.stage`.
- Using this object, you can perform various operations on the stage such as:
  - Set the title for the stage using the method **setTitle()**.

    ```
    primaryStage.setTitle("Sample application");
    ```
  - Attach the scene object to the stage using the **setScene()** method.

    ```
    primaryStage.setScene(scene);
    ```
  - Display the contents of the scene using the **show()** method as shown below.

    ```
    primaryStage.show();
    ```

Lifecycle of JavaFX Application

- The JavaFX Application class has three life cycle methods, which are:
  - `start()` – The entry point method where the JavaFX graphics code is to be written.
  - `stop()` – An empty method which can be overridden, here you can write the logic to stop the application.
  - `init()` – An empty method which can be overridden, but you cannot create stage or scene in this method.
- In addition to these, it provides a static method named `launch()` to launch JavaFX application.
- Since the `launch()` method is static, you need to call it from a static context (main generally). Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).
  - An instance of the application class is created.
  - `init()` method is called.
  - The `start()` method is called.
  - The launcher waits for the application to finish and calls the `stop()` method.

Terminating the JavaFX Application

- When the last window of the application is closed, the JavaFX application is terminated implicitly. You can turn this behavior off by passing the Boolean value “False” to the static method **`setImplicitExit()`** (should be called from a static context).
- You can terminate a JavaFX application explicitly using the methods **`Platform.exit()`** or **`System.exit(int)`**.

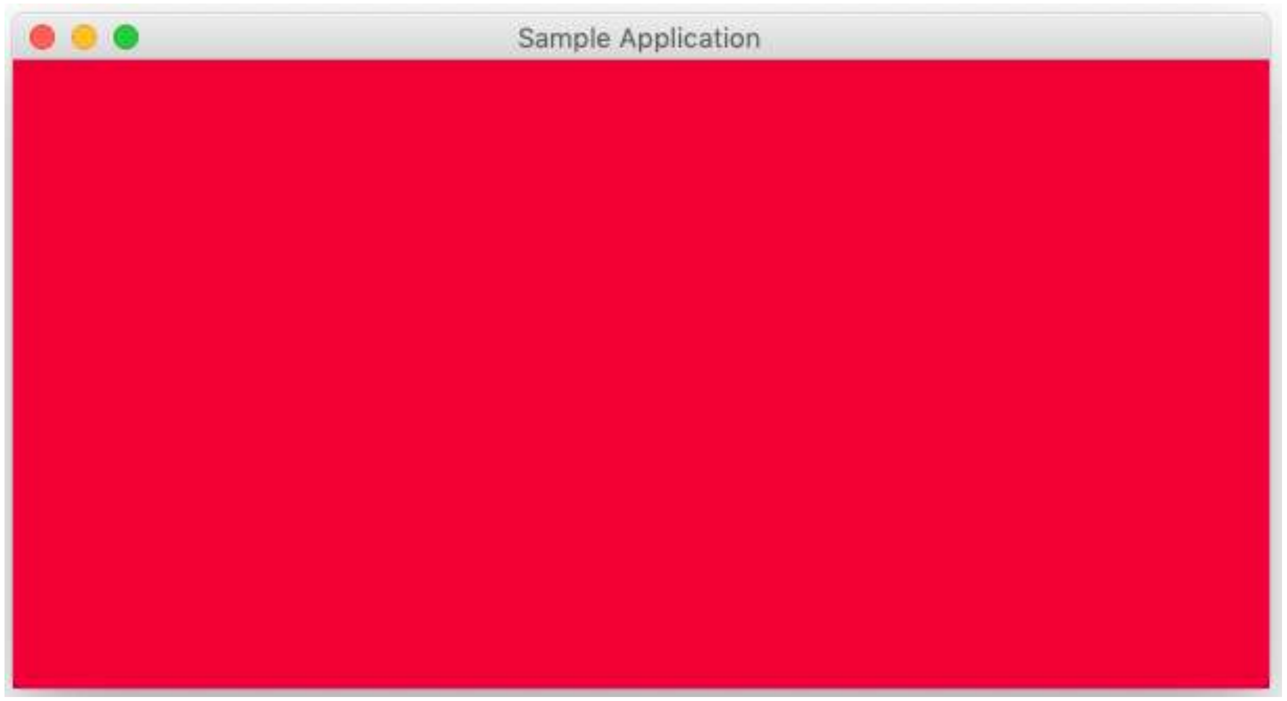
JavaFX important pakages

- The important packages of JavaFX API are:
  - **`javafx.animation:`** Contains classes to add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
  - **`javafx.application:`** Contains a set of classes responsible for the JavaFX application life cycle.
  - **`javafx.css:`** Contains classes to add CSS–like styling to JavaFX GUI applications.
  - **`javafx.event:`** Contains classes and interfaces to deliver and handle JavaFX events.
  - **`javafx.geometry:`** Contains classes to define 2D objects and perform operations on them.
  - **`javafx.stage:`** This package holds the top level container classes for JavaFX application.
  - **`javafx.scene:`** This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. There are several components that support this rich API of JavaFX.

Example – Creating an Empty Window

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavafxSample extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {

        Group group = new Group(); //creating a Group object
        Scene scene = new Scene(group ,600, 300); //Creating a Scene by passing the group object,
                                                    height and width
        scene.setFill(Color.RED); //setting color to the scene
        primaryStage.setTitle("Sample Application"); //Setting the title to Stage.
        primaryStage.setScene(scene); //Adding the scene to Stage
        primaryStage.show(); //Displaying the contents of the stage
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

Example – Creating an Empty Window (Output)



JAVAFX Nodes
-Shapes, Color, Font, Text and Image-

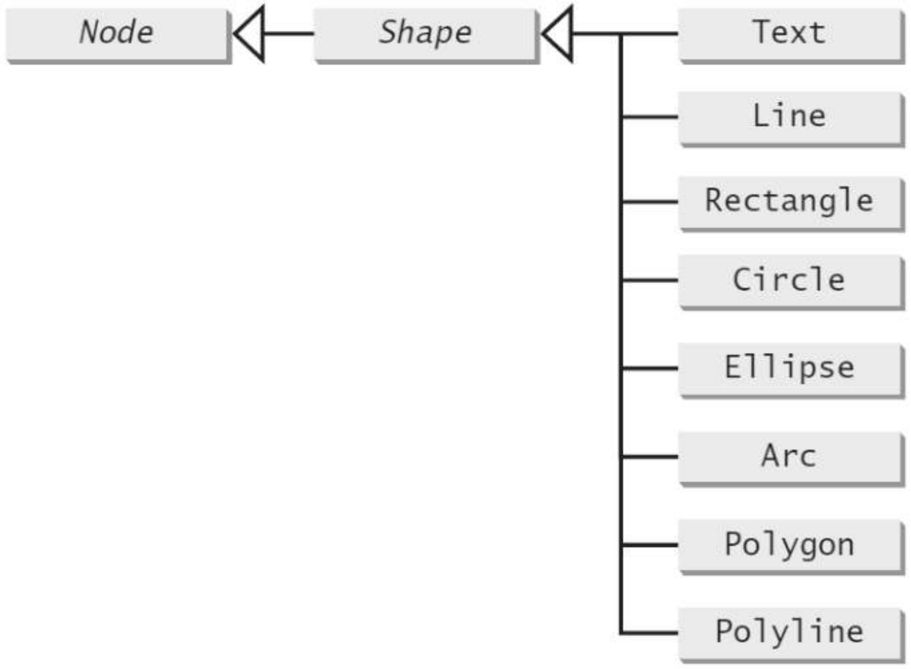
JavaFX Nodes:

JavaFX Nodes:

- **Shapes**: Text, Line, Circle, Rectangle
- **Font**
- **Colors**
- **Images**
- **Layout Panes**: Pane, StackPane, FlowPane, GridPane, BorderPane, HBox, and VBox
- **UI Controls**: Label, Button, CheckBox, RadioButton, TextField, PasswordField, TextArea, ComboBox, ListView, ScrollBar, Slider, and MediaPlayer.

JavaFX Shapes

- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.
- The `Shape` class is the abstract base class that defines the common properties for all shapes.
- Among them are the `fill`, `stroke`, and `strokeWidth` properties.
- The `fill` property specifies a color that fills the interior of a shape. The `stroke` property specifies a color that is used to draw the outline of a shape. The `strokeWidth` property specifies the width of the outline of a shape.
- The classes `Text`, `Line`, `Rectangle`, `Circle` all are subclasses of the `Shape` class. So, they inherit all the properties from the `Shape` class.

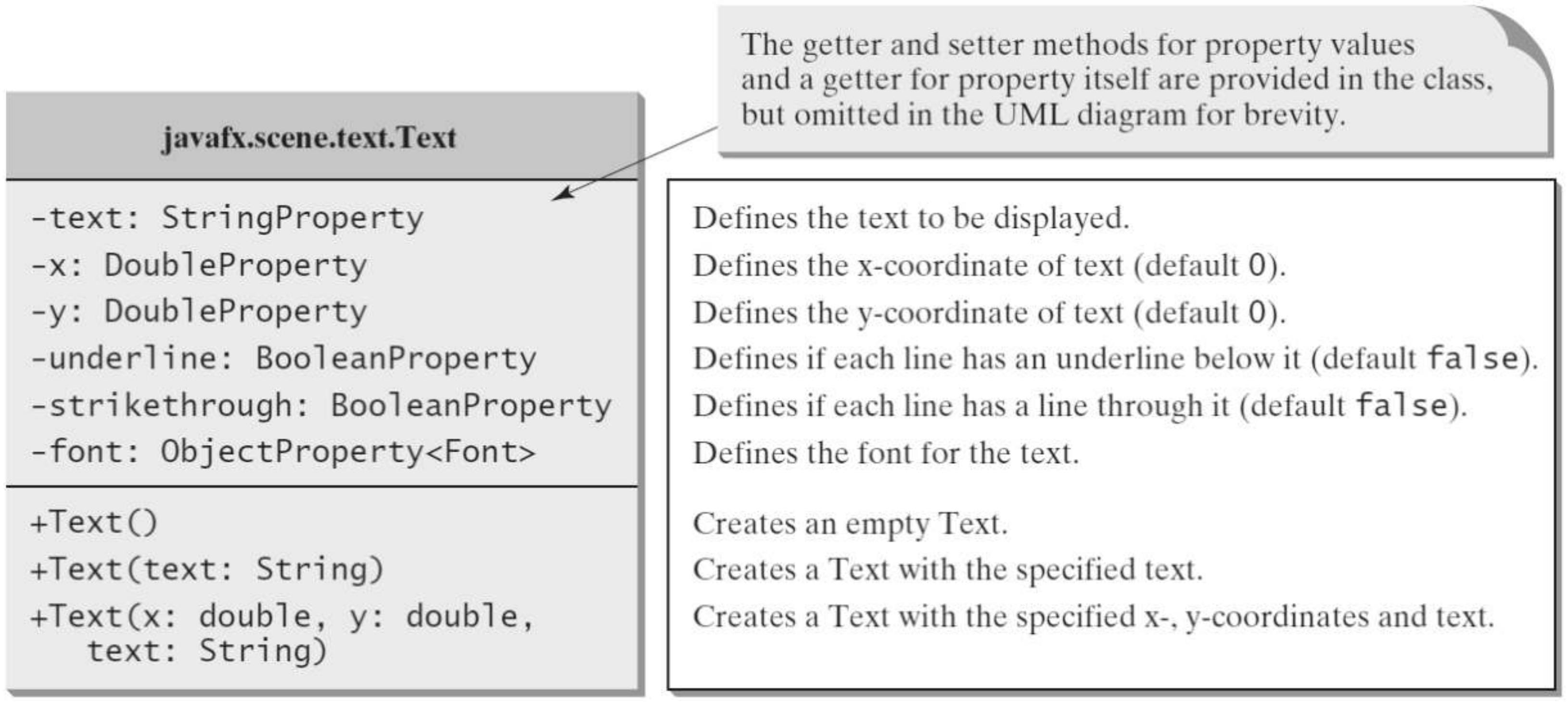


A shape is a node. The Shape class is the root of all shape classes.

JavaFX Shapes - Text

- The text node is represented by the class named **`Text`**, which belongs to the package **`javafx.scene.text`**.
- This class contains several properties to create text in JavaFX and modify its appearance. This class also inherits the **`Shape`** class which belongs to the package **`javafx.scene.shape`**.
- The class **`Text`** contains a property named **`text`** of string type, which represents the text that is to be created.
- After instantiating the **`Text`** class, you need to set value to this property using the **`setText()`** method.
- You can also set the position (origin) of the text by specifying the values to the properties x and y using their respective setter methods namely **`setX()`** and **`setY().`**
- In addition to the properties of the text like font, alignment, line spacing, text, etc. It also inherits the basic shape node properties such as **`strokeFill, stroke, strokeWidth, strokeType,`** etc.

JavaFX Shapes - Text

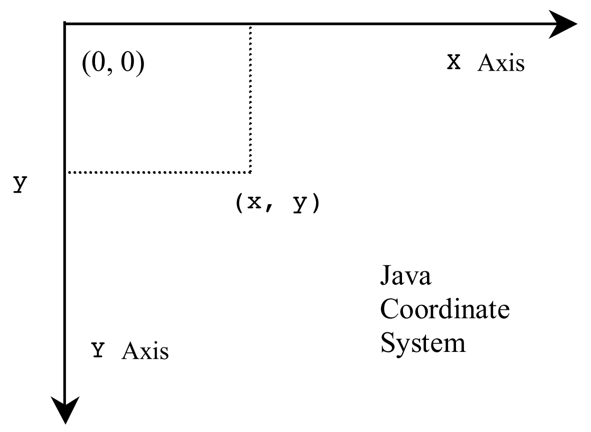


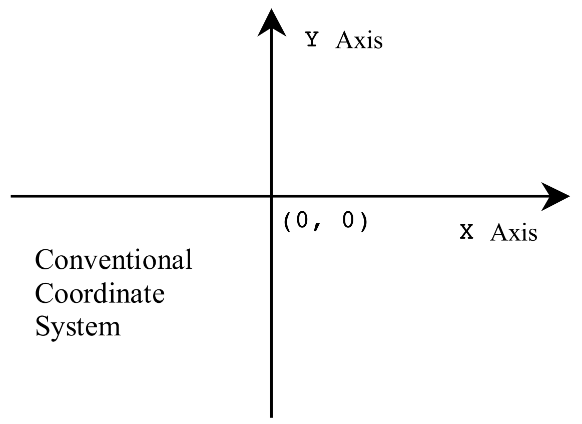
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Text	
`-text: StringProperty`	Defines the text to be displayed.
`-x: DoubleProperty`	Defines the x-coordinate of text (default `0`).
`-y: DoubleProperty`	Defines the y-coordinate of text (default `0`).
`-underline: BooleanProperty`	Defines if each line has an underline below it (default `false`).
`-strikethrough: BooleanProperty`	Defines if each line has a line through it (default `false`).
`-font: ObjectProperty<Font>`	Defines the font for the text.
`+Text()`	Creates an empty Text.
`+Text(text: String)`	Creates a Text with the specified text.
`+Text(x: double, y: double, text: String)`	Creates a Text with the specified x-, y-coordinates and text.

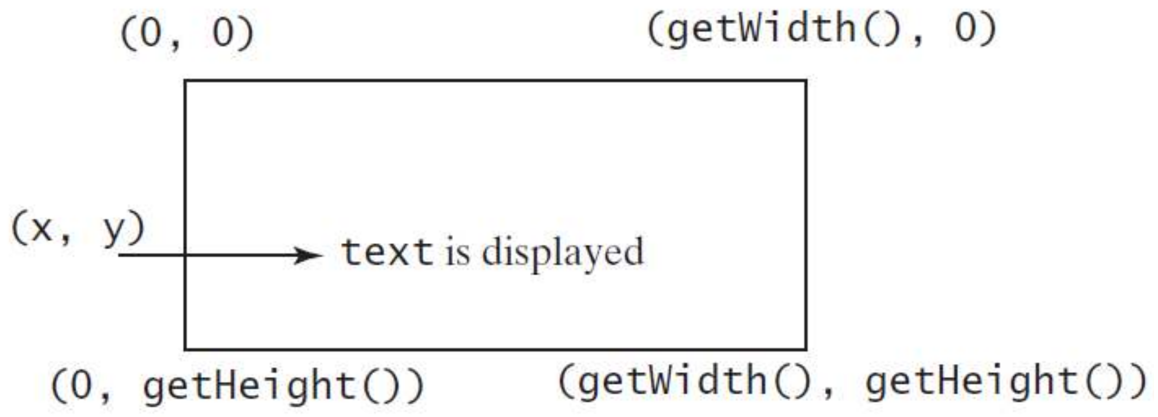
For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/text/Text.html

Java Coordinate System





Java Coordinate System

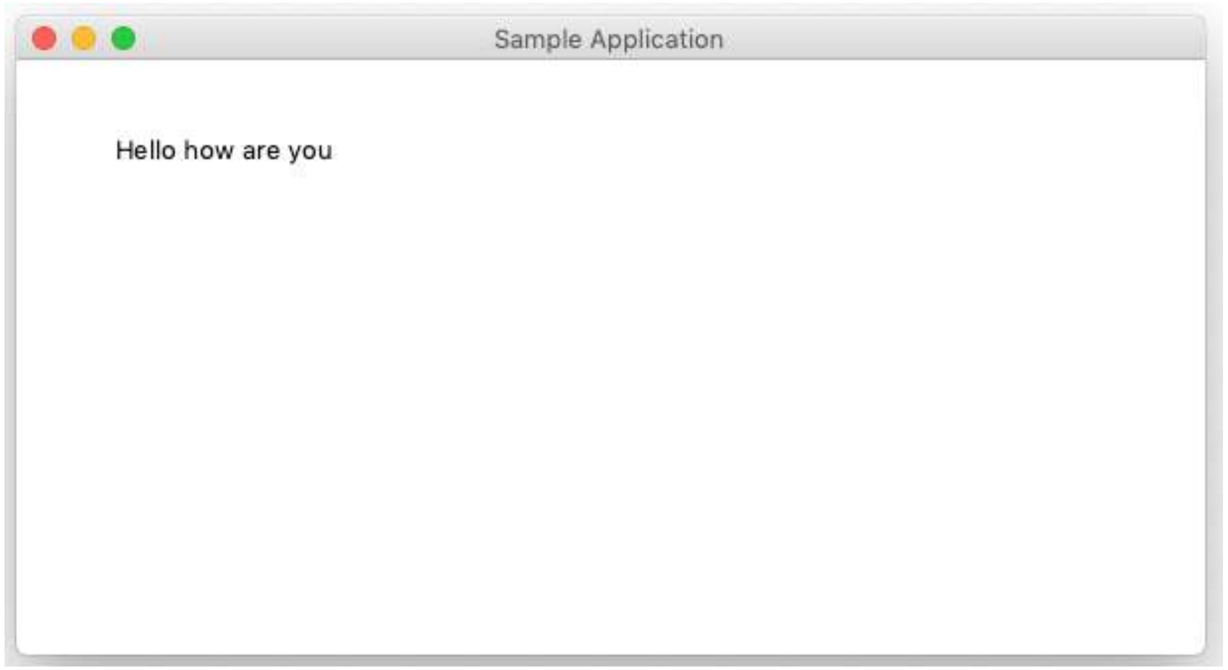


(a) Text(x, y, text)

JavaFX Shapes – Text (Creating a Text Node)

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.text.Text;

public class TextExample extends Application {
    @Override
    public void start(Stage stage) {
         Text text = new Text(); //Creating a Text object
         text.setText("Hello how are you"); //Setting the text to be added.
         text.setX(50); //setting the position of the text
         text.setY(50); //setting the position of the text
         Group root = new Group(text); //Creating a Group object
         Scene scene = new Scene(root, 600, 300); //Creating a scene object
         stage.setTitle("Sample Application"); //Setting title to the Stage
         stage.setScene(scene); //Adding scene to the stage
         stage.show(); //Displaying the contents of the stage
        }
    public static void main(String args[]){
        launch(args);
    }
}
```

JavaFX Text-Creating a Text Node



```
Hello how are you
```

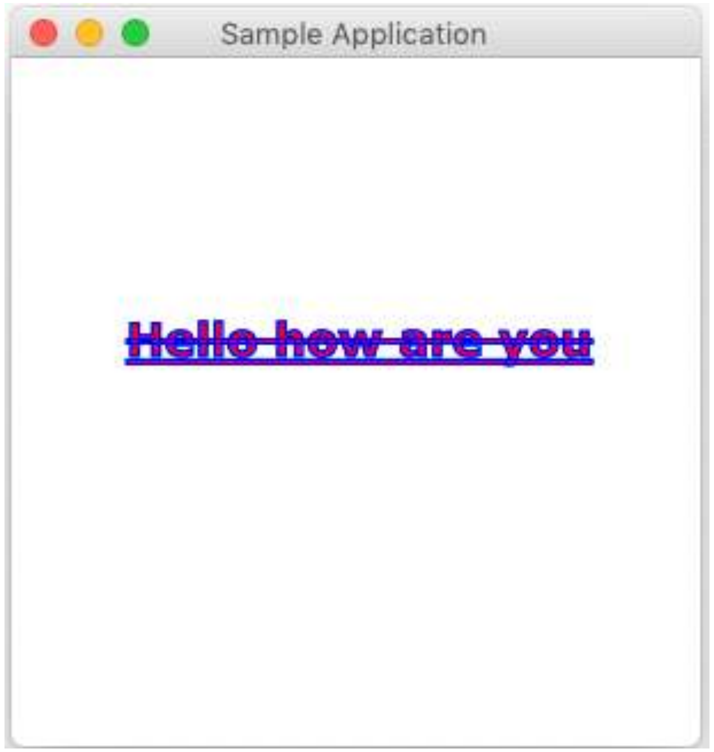
JavaFX Shapes – Text (Position, Font, Stroke, Color and decoration of the the Text)

- You can strike through the text using the method **`setStrikethrough()`**. This accepts a Boolean value, pass the value **`true`** to this method to strike through the text.
- In the same way, you can underline a text by passing the value **`true`** to the method **`setUnderLine()`**
- You can change the font size and color of the text using the **`setFont()`** method. This method accepts an object of the **`Font`** class.
- The **`Text`** class also inherits the class **`Shape`** of the package. Therefore, you can use all the method in **`javafx.scene.shape.Shape`** class to set the stroke and color to the text node too.
- You can set the color to the text using the **`setFill()`** method of the **`Shape`** (inherited) class.
- Similarly, you can set the stroke color of the text using the method **`setStroke()`**. While the width of the stroke can be set using the method **`setStrokeWidth()`**

JavaFX Shapes – Text (Position, Font, Stroke, Color and decoration of the the Text)

```java
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
...
Text text = new Text();//Creating a Text object
text.setText("Hello how are you"); //Setting the text to be added.
text.setX(50);//setting the text position
text.setY(130);

//setting the font
text.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20));

text.setFill(Color.BROWN);//Setting the color

text.setStrokeWidth(1);//setting the Stork

text.setStroke(Color.BLUE);//Setting the stroke color

text.setUnderline(true);//underlining the text

text.setStrikethrough(true);//Striking through the text
...
```

JavaFX Shapes – Text
(Position, Font, Stroke, Color and decoration of the the Text)



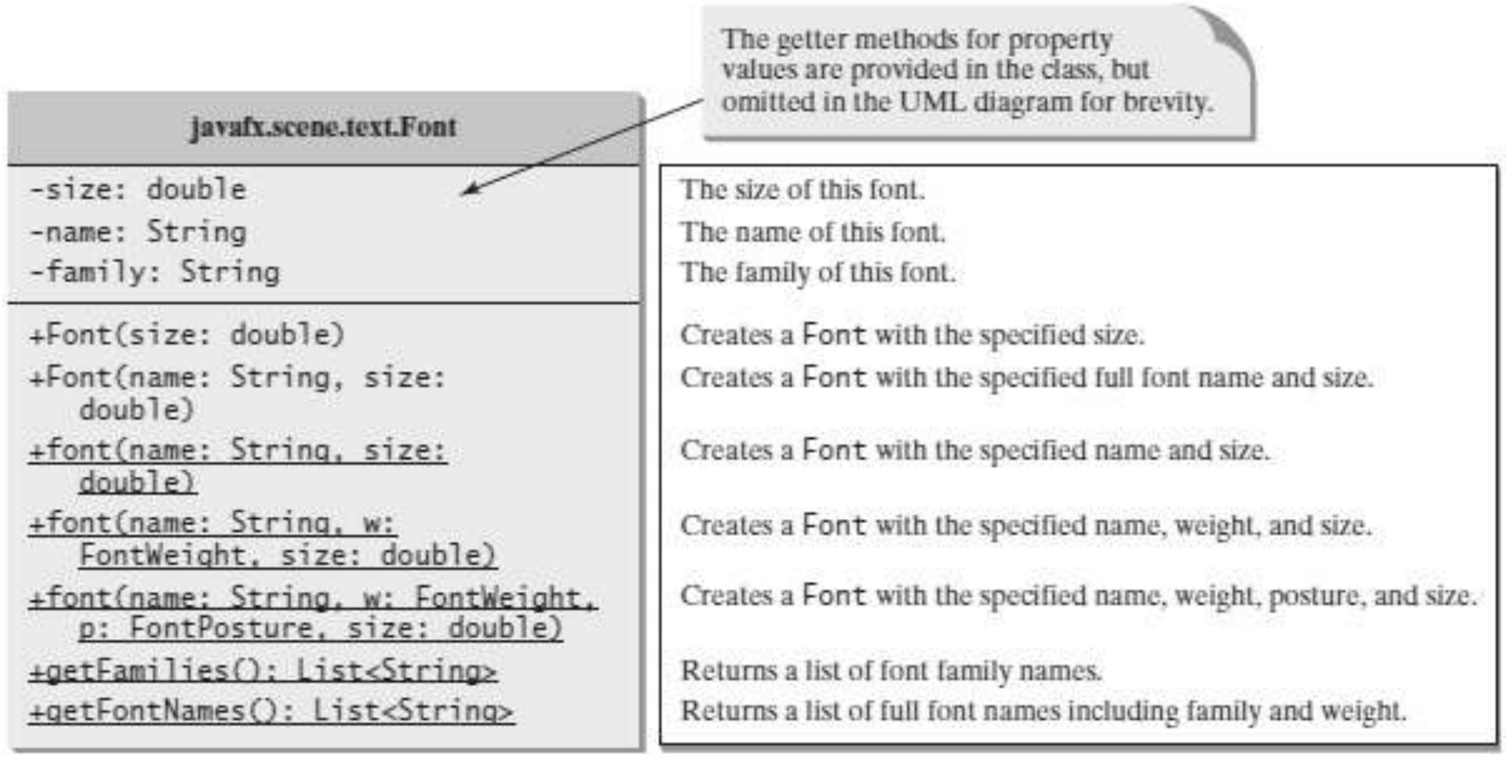
~~Hello how are you~~

JavaFX Fonts

- The class named `Font` of the package `javafx.scene.text` is used to define the font for the text. This class contains a static method named `font()`.
- This method accepts four parameters namely:
  - **family** − This is of a String type and represents the family of the font that we want to apply to the text.
  - **weight** − This property represents the weight of the font. It accepts 9 values, which are − FontWeight.BLACK, FontWeight.BOLD, FontWeight.EXTRA_BOLD, FontWeight.EXTRA_LIGHT, LIGHT, MEDIUM, NORMAL, SEMI_BOLD, THIN.
  - **posture** − This property represents the font posture (regular or italic). It accepts two values FontPosture.REGULAR and FontPosture.ITALIC.
  - **size** − This property is of type double and it represents the size of the font.

```
text.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20));
```

JavaFX Fonts

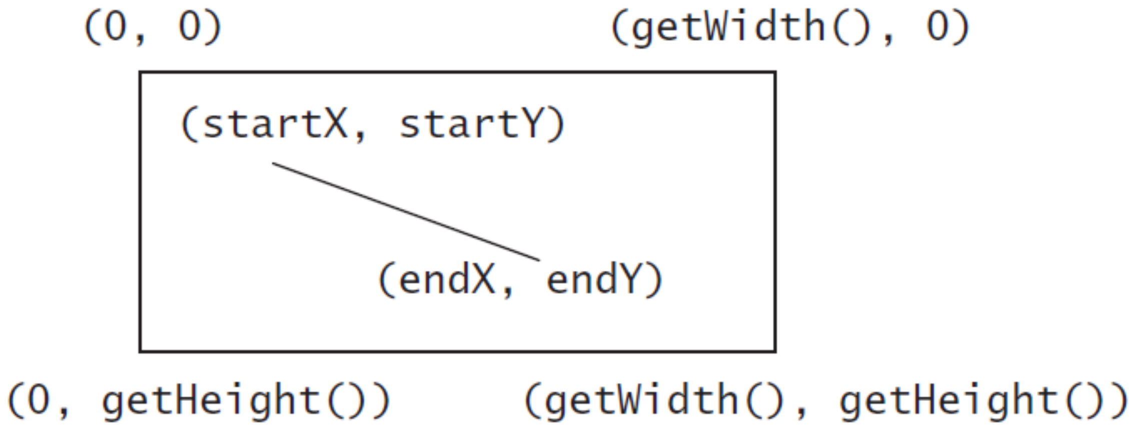


The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

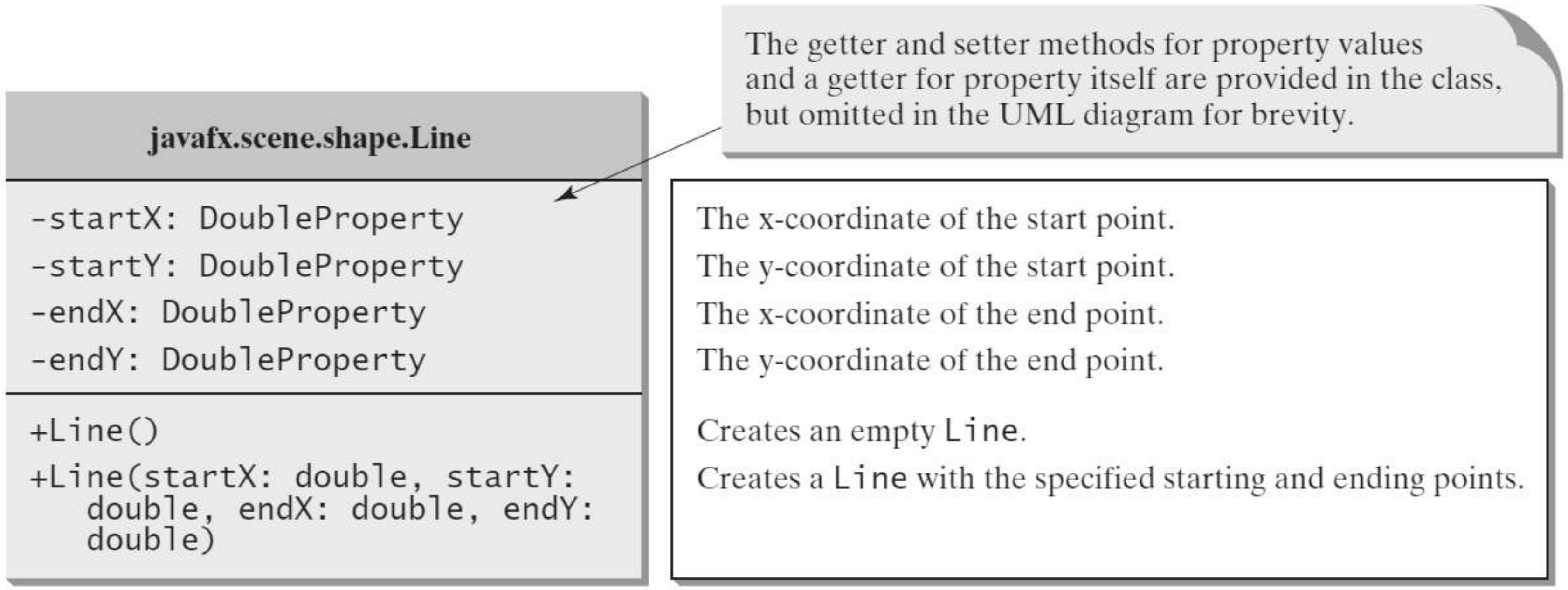
javafx.scene.text.Font	
-size: double	The size of this font.
-name: String	The name of this font.
-family: String	The family of this font.
+Font(size: double)	Creates a Font with the specified size.
+Font(name: String, size: double)	Creates a Font with the specified full font name and size.
+font(name: 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.

For full properties and method description: https://docs.oracle.com/javafx/2/api/javafx/scene/text/Font.html

JavaFX Shapes – Line



JavaFX Shapes – 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	The x-coordinate of the start point.
-startY: DoubleProperty	The y-coordinate of the start point.
-endX: DoubleProperty	The x-coordinate of the end point.
-endY: DoubleProperty	The y-coordinate of the end point.
+Line()	Creates an empty Line.
+Line(startX: double, startY: double, endX: double, endY: double)	Creates a Line with the specified starting and ending points.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Line.html

JavaFX Shapes – Line (Creating a Line Node)

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Line;
import javafx.stage.Stage;

public class LineExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        Line line = new Line();
        line.setStartX(100.0f); //Setting the Properties of the Line
        line.setStartY(140.0f);
        line.setEndX(300.0f);
        line.setEndY(140.0f);
        Group root = new Group(line);
        Scene scene = new Scene(root, 400, 300);
        primaryStage.setTitle("Drawing Line"); // 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);}
```

Note that:
`Line` class inherits the Shape class which belongs to the package:
`javafx.scene.shape`

For that, we can set the color to the `Line` using the method:

`setFill()`

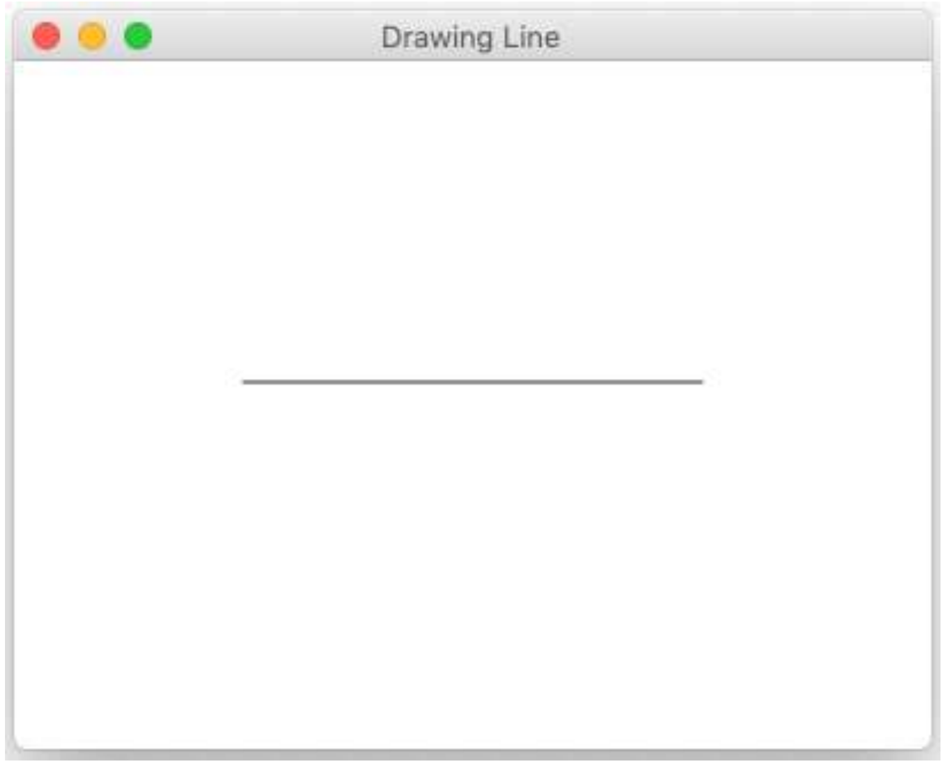
Similarly, we can set the stroke color of the `Line` using the method:

`setStroke()`

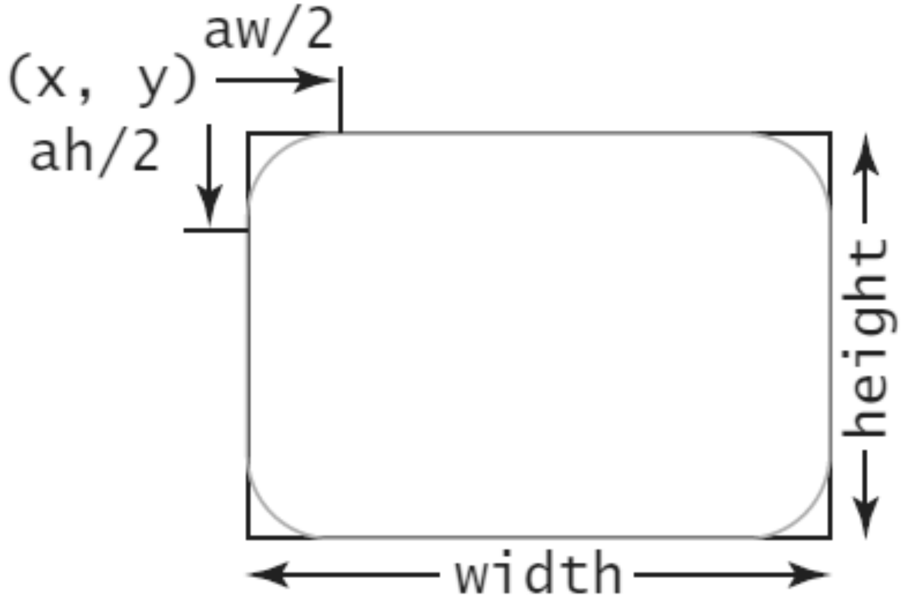
While the width of the stroke can be set using the method:

`setStrokeWidth()`

JavaFX Shapes – Line (Creating a Line Node)

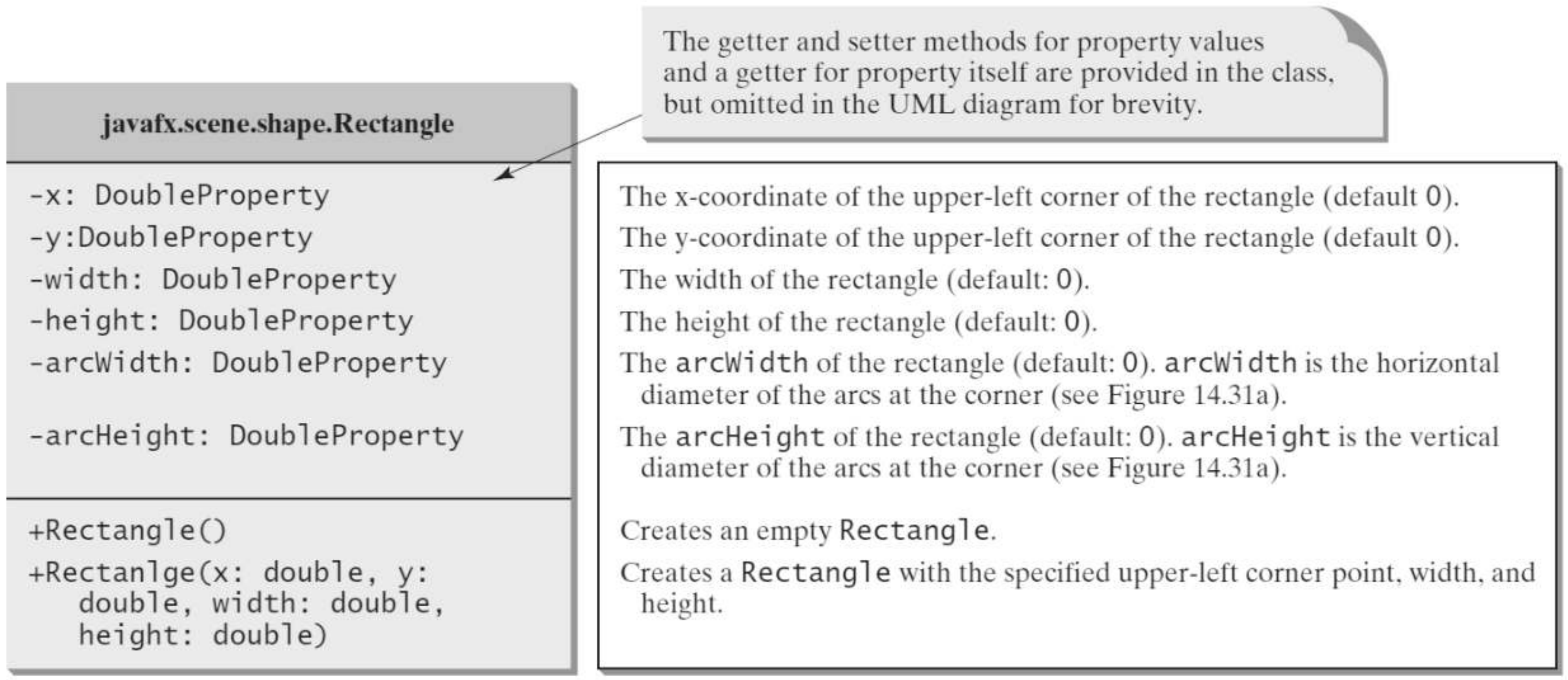


JavaFX Shapes – Rectangle



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

JavaFX Shapes – 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`	The x-coordinate of the upper-left corner of the rectangle (default 0).
`-y:DoubleProperty`	The y-coordinate of the upper-left corner of the rectangle (default 0).
`-width: DoubleProperty`	The width of the rectangle (default: 0).
`-height: DoubleProperty`	The height of the rectangle (default: 0).
`-arcWidth: DoubleProperty`	The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).
`-arcHeight: DoubleProperty`	The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).
`+Rectangle()`	Creates an empty `Rectangle`.
`+Rectanlge(x: double, y: double, width: double, height: double)`	Creates a `Rectangle` with the specified upper-left corner point, width, and height.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Rectangle.html

JavaFX Shapes – Rectangle (Creating a Rectangle Node)

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class RectangleExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        Rectangle rectangle = new Rectangle();
        rectangle.setX(100); //Setting the Properties of the Rectangle
        rectangle.setY(100);
        rectangle.setWidth(200);
        rectangle.setHeight(100);
        Group root = new Group(rectangle);
        Scene scene = new Scene(root, 400, 300);
        primaryStage.setTitle("Drawing Rectangle"); // 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);}
```

Note that:
`Rectangle` class inherits the Shape class which belongs to the package:
`javafx.scene.shape`

For that, we can set the color to the `Rectangle` using the method:

`setFill()`

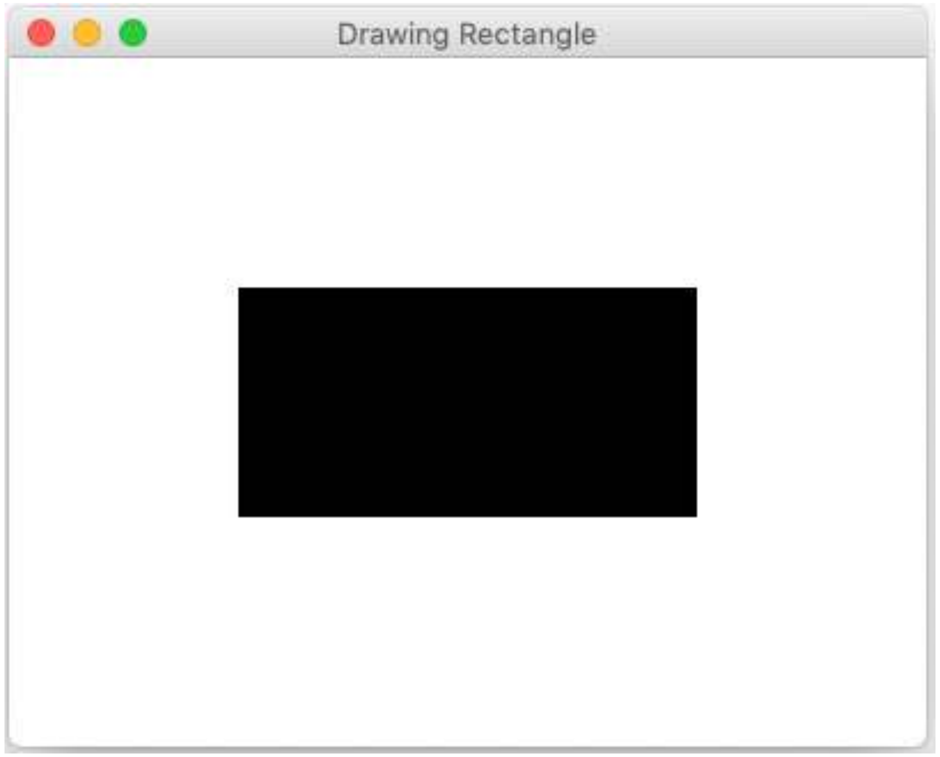
Similarly, we can set the stroke color of the `Rectangle` using the method:

`setStroke()`

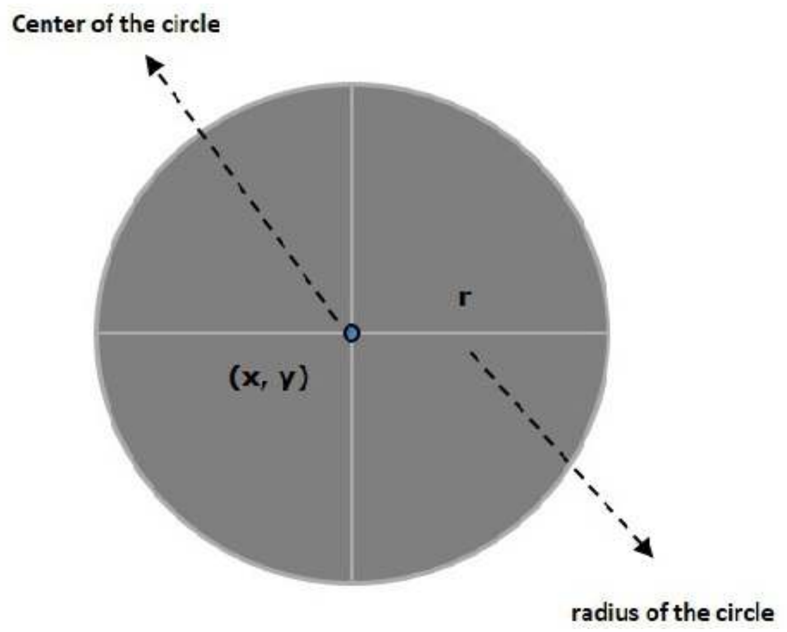
While the width of the stroke can be set using the method:

`setStrokeWidth()`

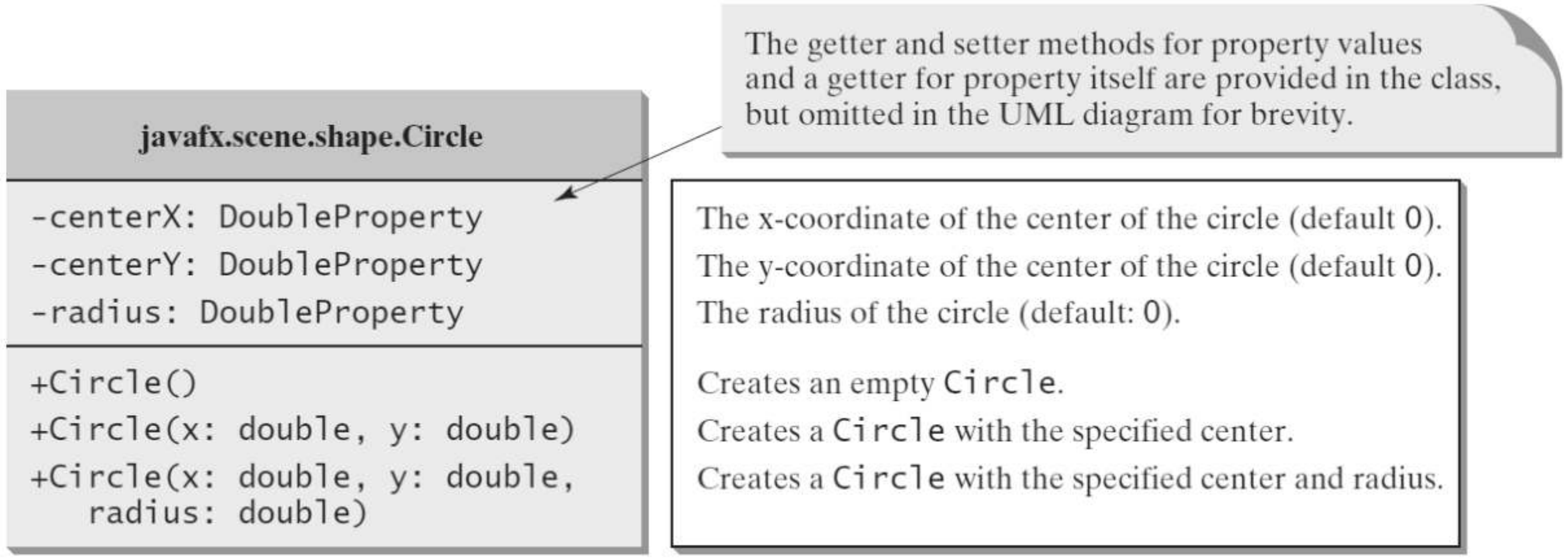
JavaFX Shapes – Rectangle (Creating a Rectangle Node)



JavaFX Shapes – Circle



JavaFX Shapes – 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`	The x-coordinate of the center of the circle (default 0).
`-centerY: DoubleProperty`	The y-coordinate of the center of the circle (default 0).
`-radius: DoubleProperty`	The radius of the circle (default: 0).
`+Circle()`	Creates an empty `Circle`.
`+Circle(x: double, y: double)`	Creates a `Circle` with the specified center.
`+Circle(x: double, y: double, radius: double)`	Creates a `Circle` with the specified center and radius.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Circle.html

JavaFX Shapes – Circle (Creating a Circle Node)

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class CircleExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        Circle circle = new Circle();
        circle.setCenterX(200); //Setting the Properties of the Circle
        circle.setCenterY(150);
        circle.setRadius(50);
        Group root = new Group(circle);
        Scene scene = new Scene(root, 400, 300);
        primaryStage.setTitle("Drawing Rectangle"); // 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);}
```

Note that:
`Circle` class inherits the Shape class which belongs to the package:
`javafx.scene.shape`

For that, we can set the color to the `Circle` using the method:

`setFill()`

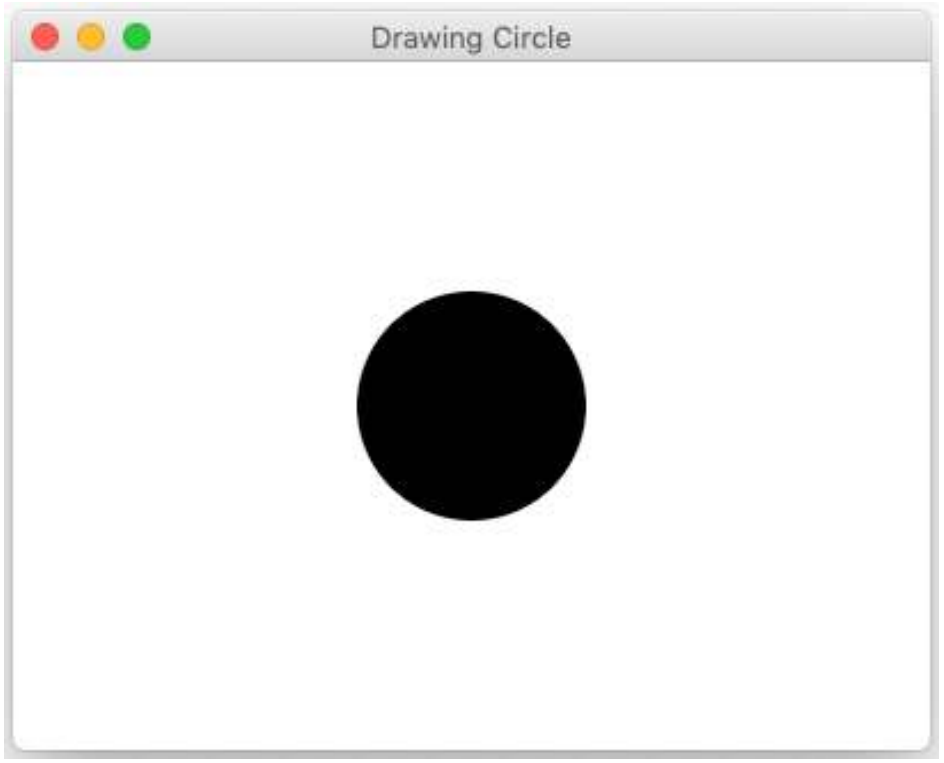
Similarly, we can set the stroke color of the `Circle` using the method:

`setStroke()`

While the width of the stroke can be set using the method:

`setStrokeWidth()`

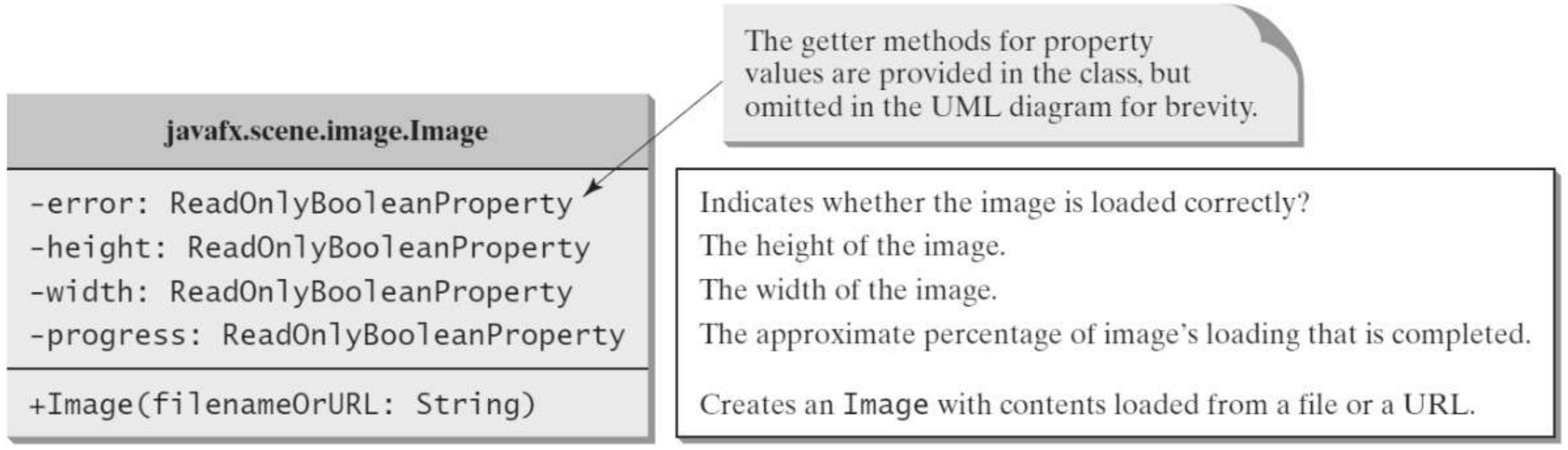
JavaFX Shapes – Circle (Creating a Circle Node)



JavaFX Image

- You can load and modify images using the classes provided by JavaFX in the package `javafx.scene.image`. JavaFX supports the image formats like **Bmp, Gif, Jpeg, Png**.
- You can load an image in JavaFX by instantiating the class named `Image` of the package `javafx.scene.image`.
- To the constructor of the class, you have to pass either of the following:
  - An `InputStream` object of the image to be loaded or,
  - A string variable holding the URL for the image.
- After loading the image, you can set the view for the image by instantiating the `ImageView` class and passing the image to its constructor.

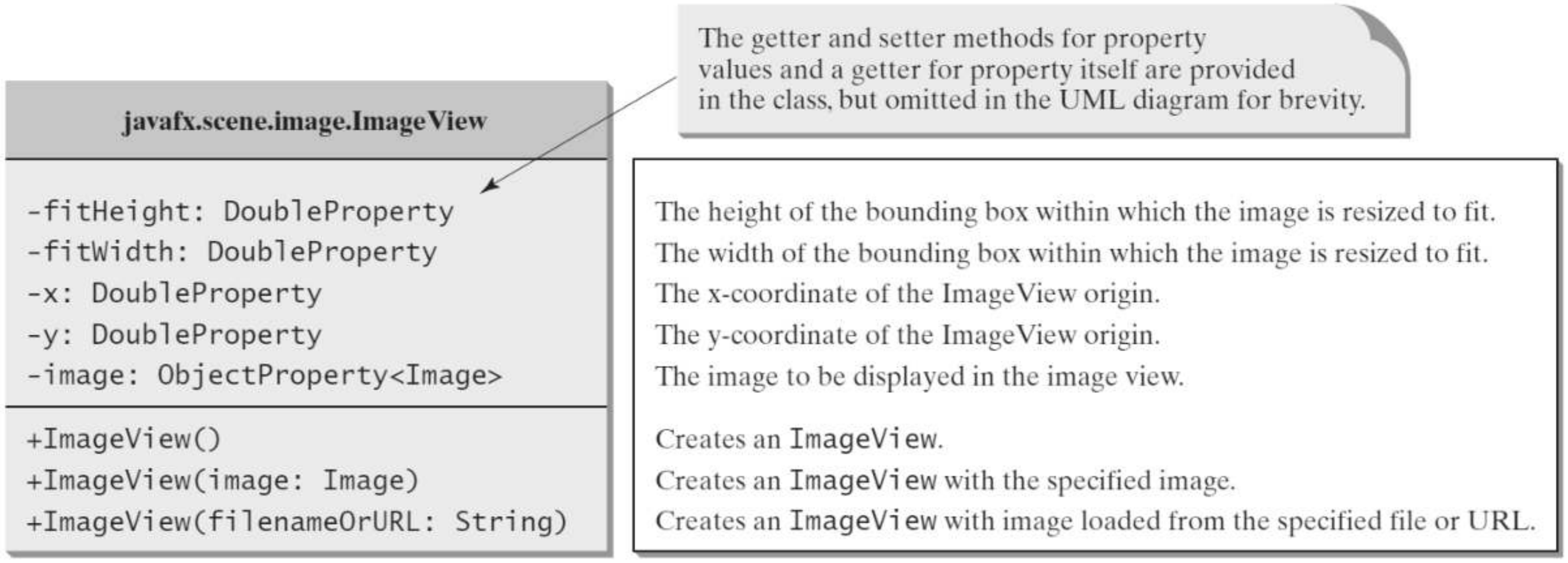
The Image Class


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	Indicates whether the image is loaded correctly?
-height: ReadOnlyBooleanProperty	The height of the image.
-width: ReadOnlyBooleanProperty	The width of the image.
-progress: ReadOnlyBooleanProperty	The approximate percentage of image's loading that is completed.
+Image(filenameOrURL: String)	Creates an Image with contents loaded from a file or a URL.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html

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.

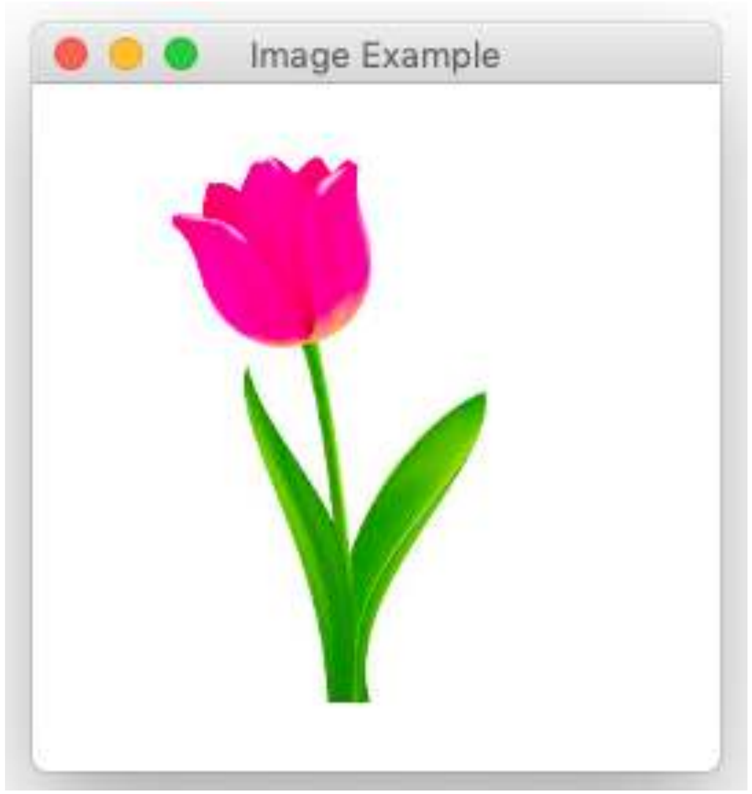
Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html

JavaFX Image

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
public class ImageExample extends Application {
    @Override
    public void start(Stage stage){
        Image image = new Image("https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcQs1XBoCplUS1v2gkTinxGVh20QnCy-DWy8lyQDTvt-ldf0WINZ");
        ImageView imageView = new ImageView(image);//Setting the image view
        imageView.setX(50); //Setting the image position
        imageView.setY(25);
        imageView.setFitHeight(200); //setting the fit height and width of the image view
        imageView.setFitWidth(200);
        imageView.setPreserveRatio(true);  //Setting the preserve ratio of the image view
        Group root = new Group(imageView); //Creating a Group object
        Scene scene = new Scene(root, 250, 250);//Creating a scene object
        stage.setTitle("Image Example"); //Setting title to the Stage
        stage.setScene(scene);  //Adding scene to the stage
        stage.show();//Displaying the contents of the stage
    }
public static void main(String args[]){launch(args);}}
```

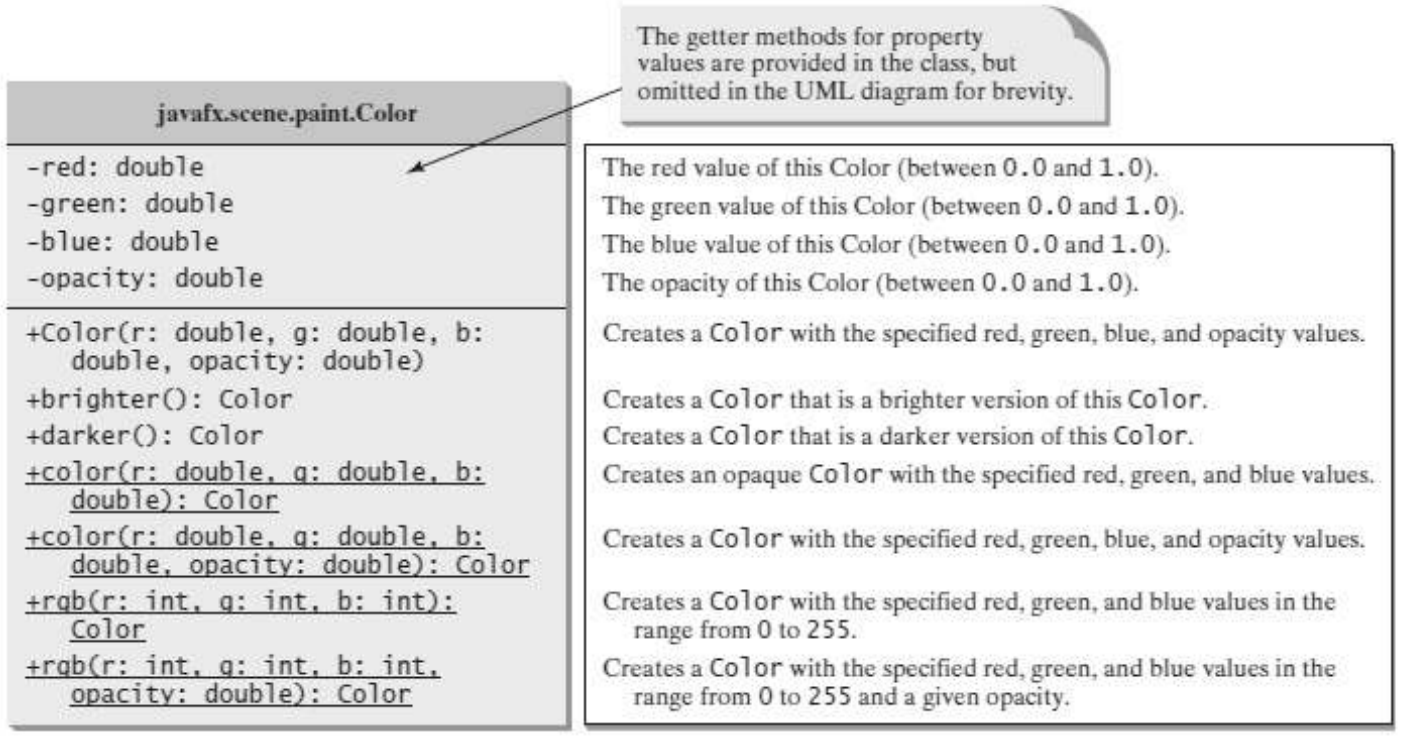
JavaFX Image



JavaFX Colors

- To apply colors to an application, JavaFX provides various classes in the package `javafx.scene.paint` package. This package contains an abstract class named Paint and it is the base class of all the classes that are used to apply colors.
- Using these classes, you can apply colors in the following patterns −
  - **Uniform** − In this pattern, color is applied uniformly throughout node.
  - **Image Pattern** − This lets you to fill the region of the node with an image pattern.
  - **Gradient** − In this pattern, the color applied to the node varies from one point to the other. It has two kinds of gradients namely **Linear Gradient** and **Radial Gradient**.
- All those node classes to which you can apply color such as **Shape** and **Text** (including Scene), have methods named `setFill()` and `setStroke()`. These will help to set the color values of the nodes and their strokes respectively.

JavaFX Colors



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.
<u>+color(r: double, g: double, b: double): Color</u>	Creates an opaque Color with the specified red, green, and blue values.
<u>+color(r: double, g: double, b: double, opacity: double): Color</u>	Creates a Color with the specified red, green, blue, and opacity values.
<u>+rgb(r: int, g: int, b: int): Color</u>	Creates a Color with the specified red, green, and blue values in the range from 0 to 255.
<u>+rgb(r: int, g: int, b: int, opacity: double): Color</u>	Creates a Color with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

For full properties and method description: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html

JavaFX Colors
- Applying Color to the Nodes

- To set uniform color pattern to the nodes, you need to pass an object of the class color to the `setFill()`, `setStroke()` methods as follows:

```
//Setting color to the text
Color color = new Color.BEIGE
text.setFill(color);

//Setting color to the stroke
Color color = new Color.DARKSLATEBLUE
circle.setStroke(color);
```

- In the above code block, we are using the static variables of the color class to create a color object.

JavaFX Colors
- Applying Color to the Nodes

- In the same way, you can also use the RGB values or HSB standard of coloring or web hash codes of colors.
- Example:

```
//creating color object by passing RGB values
Color c = Color.rgb(0,0,255);

//creating color object by passing HSB values
Color c = Color.hsb(270,1.0,1.0);

//creating color object by passing the hash code for web
Color c = Color.web("0x0000FF",1.0);
```

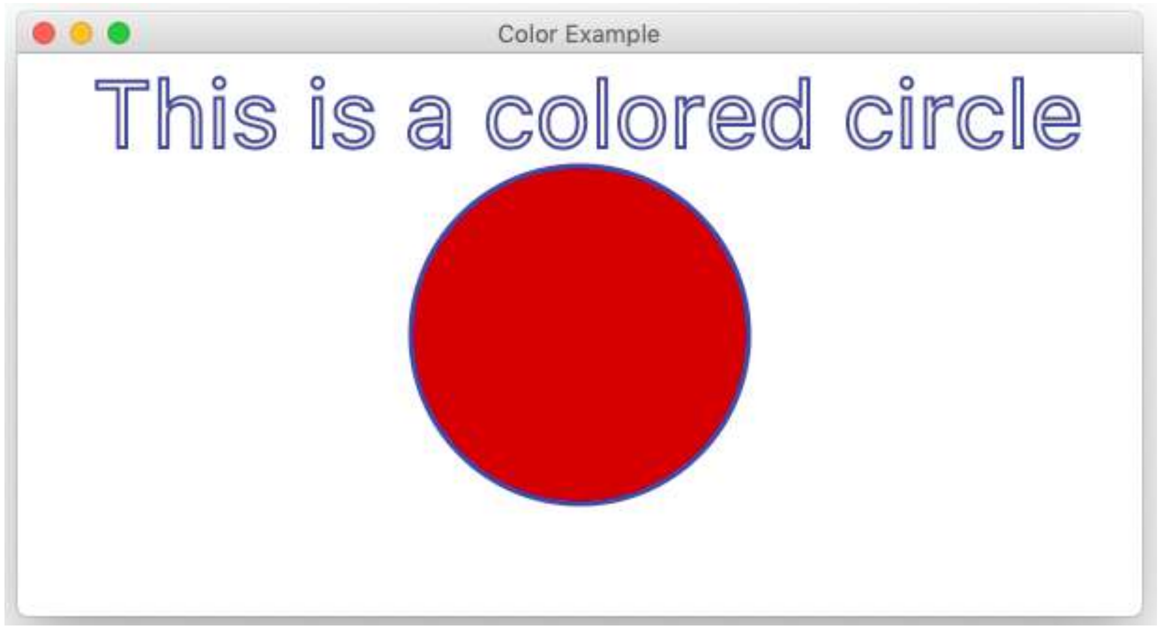
JavaFX Colors
- Applying Color to the Nodes (Example)

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Circle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

public class ColorExample extends Application {
    @Override
    public void start(Stage stage) {
        Circle circle = new Circle();//Drawing a Circle
        circle.setCenterX(300.0f); //Setting the properties of the circle
        circle.setCenterY(150.0f);
        circle.setRadius(90.0f);
        circle.setFill(Color.DARKRED);//Setting color to the circle
        circle.setStrokeWidth(3); //Setting the stroke width
        circle.setStroke(Color.DARKSLATEBLUE); //Setting color to the stroke
         .
         .
         .
```

JavaFX Colors
- Applying Color to the Nodes (Example)

```
        Text text = new Text("This is a colored circle");//Drawing a text
        text.setFont(Font.font("Edwardian Script ITC", 50));//Setting the font of the text
        text.setX(40);//Setting the position of the text
        text.setY(50);
        text.setFill(Color.BEIGE);//Setting color to the text
        text.setStrokeWidth(2);
        text.setStroke(Color.DARKSLATEBLUE);
        Group root = new Group(circle, text); //Creating a Group object
        Scene scene = new Scene(root, 600, 300); //Creating a scene object
        stage.setTitle("Color Example"); //Setting title to the Stage
        stage.setScene(scene);  //Adding scene to the stage
        stage.show();//Displaying the contents of the stage
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

JavaFX Colors
- Applying Color to the Nodes (Example)



JavaFX Colors
- Applying Image Pattern to the Nodes

- To apply an image pattern to the nodes, instantiate the `ImagePattern` class and pass its object to the `setFill()`, `setStroke()` methods.
- The constructor of this class accepts six parameters namely:
  - Image – The object of the image using which you want to create the pattern.
  - x and y – Double variables representing the (x, y) coordinates of origin of the anchor rectangle.
  - height and width – Double variables representing the height and width of the image that is used to create a pattern.
  - isProportional – This is a Boolean Variable; on setting this property to true, the start and end locations are set to be proportional.
  - Example:

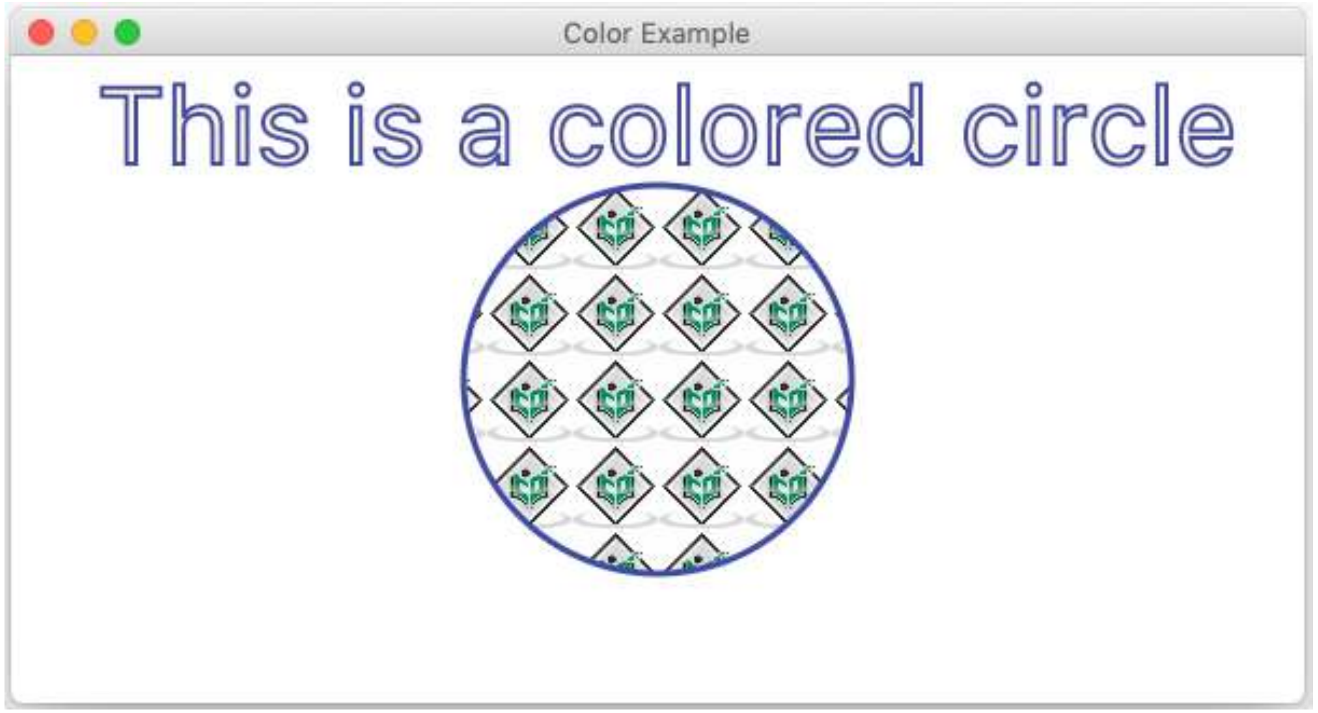
```
ImagePattern example = new ImagePattern(URL, 20, 20, 40, 40, false);
```

JavaFX Colors
- Applying Image Pattern to the Nodes Example

- Same as the previous example except for what you use as parameters in `setFill()` method.

```
//Setting the image pattern
String link ="https://encrypted-
tbn1.gstatic.com/images?q=tbn:ANd9GcRQub4GvEezKMsiIf67UrOxSzQuQ9zl5ysnjRn87VOC8tAdgmAJjcwZ2q
M";

Image image = new Image(link);

ImagePattern pattern = new ImagePattern(image,20,20,40,40,false);

//Setting the image pattern to the circle and text
circle.setFill(pattern);

text.setFill(pattern);
```

JavaFX Colors
- Applying Image Pattern to the Nodes Example



This is a colored circle