JavaFX Architecture

- Javafx.animation – add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.

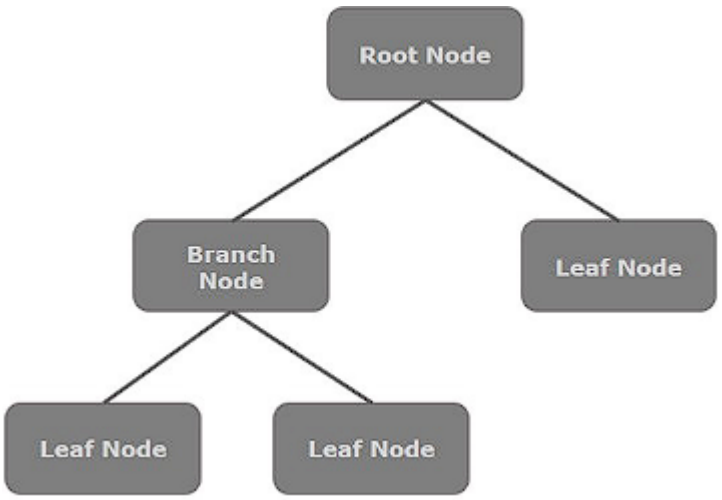
javafx. …	Information
animation	add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
application	responsible for the JavaFX application life cycle.
css	add CSS–like styling to JavaFX GUI applications.
event	Classes and interfaces to deliver and handle javafx events
geometry	Contains classes to define 2D objects and perform operations on them
stage	Holds the top level container classes for the javaFX application
scene	Classes and interfaces to support the scene graph. Additionally provides sub-packages – canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web

Scene Graph

- The starting point of the construction of the GUI application.
- In JavaFX, the GUI applications were coded using a scene graph.
- Holds the (GUI) application primitive that are termed as nodes.

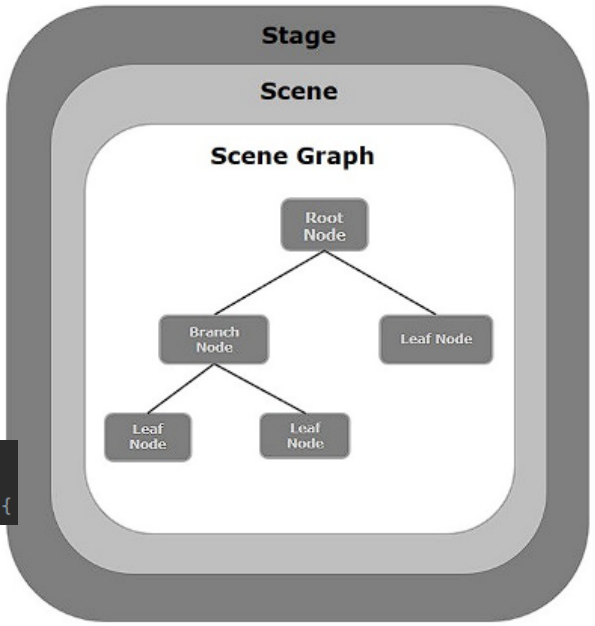
Nodes

- A visual/graphical object
- may include:
    - Geometrical objects – 2D and 3D, such as circle, rectangle etc
    - UI controls – Buttons, Checkbox, Choice box, Text Area.
    - Containers – (layout panes) such as Border Pane, Grid Pane, Flow Pane
    - Media elements – such as audio, video and image objects/
- In general a collection of them makes a scene graph and are arranged in a hierarchical order:



JavaFX application structure

- Stage
- Structure
- Nodes



Stage

- A window.
- Contains all objects of the JavaFX application.
- The primary stage is created by the platform itself.
- ^ and passed as an argument to the start() method of the Application class.

```
public class Main extends Application {

    public void start(Stage primaryStage) throws Exception {
```

- Call <stage>**.show()** to display the stage.

Scene

- Contains all the contents of a scene graph. (nodes)
- Scene scene = new Scene(<root node>)

Scene Graph and Nodes

- **Scene graph** – tree-like data structure representing the contents of a scene.
- **Node** –
    - a visual/graphical object of a scene graph
    - 3 Types
        - Root Node – first Scene Graph Node
        - Branch/Parent Node –
            - Has child nodes.
            - Abstract class ‘Parent’ (javafx.scene) is the base class of all parent nodes.
        - Leaf Node – Zero child nodes e.g. Rectangle, Box, ImageView etc

Parent Node Types

- **Group** – a collective node containing a list of children nodes that are rendered in order. Any transformation affects all children.
- **Region** – Base class of all JavaFX Node based UI controls such as Chart, Pane and Control.
- **WebView** – Manages the web engine and displays its contents.

Creating a JavaFX Application

- Instantiate the Application class (by extending it to your javafx classes)
- Implementing its method start(), writing your graphics code in here.
- In the main method, launch the application using launch()

```
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);
   }
}
```

Start method

- Prepare a scene graph with the required nodes.
- Prepare a scene with the required dimensions and add the scene graph to it.
- Prepare a stage and add the scene to the stage.
- Display the contents of the stage.

Preparing the Scene Graph

- You need to create a root node – Group, Region or WebView
- Group root = new Group();
- ObservableList list = root.getChildren();
  list.add(<nodeObject>);
- Group root = new Group(NodeObject);

Region

- It is the Base class of all the JavaFX Node-based UI Controls
- **Chart** – embeds charts in your application
    - Types
        - PieChart
        - XYChart
            - AreaChart
            - BarChart
            - BubbleChart
            - LineChart
            - ScatterChart

            - StackedAreaChart
            - StackedBarChart
        - Creating a chart
            - Defining the axis
            - Instantiate the respective class.
            - Prepare and pass data to the chart.

- **Pane**
    - After adding nodes to a scene, we generally arrange them in order. This arrangement within the container is called the Layout (or *Pane*)
    - **HBox** – horizontal row of nodes.
    - **VBox** – vertical column of nodes.
    - **BorderPane** – arranges nodes in top, left, bottom and centre positions.
    - **StackPane** – put nodes on top eachother like a stack, first bottom, last top.
    - **TextFlow** – arranges multiple text nodes
    - **AnchorPane** – layout anchors the nodes at a particular distance from the pane.
    - **TilePane** – lays out nodes in the form of uniformly sized tiles.
    - **GridPane** – lays out nodes as a grid of rows and columns.
    - **FlowPane** – Wraps all nodes in a flow. Horizontal - wraps nodes at its height. Vertical – width.

- Control
    - Base class of UI controls such as
    - **Label** – places text
    - **Button** – labelled button
    - **ColorPicker** – provides a pane of controls designed to allow a user to manipulate and select a colour.
    - **CheckBox** – true or false state.
    - **RadioButton** – true or false state in a group.
    - **ListView** – scrolling list of text items
    - **TextField** – editable single line of text component.
    - **PasswordField** – text component specialised for password entry.
    - **Scrollbar** – scrollbar component enabling user to select from a range of values
    - **FileChooser** – dialog window from which the user can select a file.
    - **ProgressBar** – as the task progresses towards completion, bar displays percentage of progress.
    - **Slider** – Graphically select a value by sliding a knob within an interval
    - Accordion
    - ButtonBar
    - ChoiceBox
    - ComboBoxBase
    - HTMLEditor etc

Preparing the scene

- Scene s = new Scene(<root node> [, <width>, <height>]);
- Scene s = new Scene(root); Scene s = new Scene(root, 600, 300);

Preparing the Stage

- <stage>.setTitle("..") – sets the title of the window/stage.
- <stage>.setScene(<Scene>) – sets the scene to the stage.
- <stage>.show() – displays the stage

Lifecycle of JavaFx Application

- **start()** –
    - Entry point method where graphics code is to be written.
- **stop()** –
    - Empty method which can be overwritten.
    - Can write logic to stop the application.
- **init()** –
    - Empty method which can be overwritten.
    - Cannot create a stage or scene.
- **launch()** –
    - Additional static method to launch the JavaFX method.
    - Since static, launch from static context (main mostly)
    - Whenever an app is launched, the following occurs
        1. An instance of the application class is created.
        2. Init() method called.
        3. start() method called.
        4. Waits for the application to finish and calls the stop() method.

Example Application

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

public class DrawingLine extends Application{
   @Override
   public void start(Stage stage) {
      //Creating a line object
      Line line = new Line();

      //Setting the properties to a line
      line.setStartX(100.0);
      line.setStartY(150.0);
      line.setEndX(500.0);
      line.setEndY(150.0);

      //Creating a Group
      Group root = new Group(line);

      //Creating a Scene
      Scene scene = new Scene(root, 600, 300);

      //Setting title to the scene
      stage.setTitle("Sample application");

      //Adding the scene to the stage
      stage.setScene(scene);

      //Displaying the contents of a scene
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

More on stages

- Image icon = new Image("<image file name>")
  stage.getIcons().add(icon) – adds an icon to the stage's window.
- stage.setWidth(<integer>) – changes window size
- stage.setX(<int>) – changes window position.
- stage.setFullScreen(<bool>) – makes the window fullscreen.
- stage.setFullScreenExitHint("<hint to which button to press>")
- stage.setFullScreenExitKeyCombination(KeyCombination.valueOf("<key>"))

More on scenes

- Group root = new Group();
- Scene s = new Scene(root,<width>,<height>,Color.<color>)

- Text t = new Text();
- t.setText("<text>"); t.setX(0); t.setY(0);
- t.setFont("Font.font("<font name>"), <size>);
- t.setFill(Color.<colour>)
- root.getChildren().add(t);

- Line line = new Line();
- Rectangle rectangle = new Rectangle();
- Polygon triangle = new Polygon();
  triangle.getPoints().setAll(
  200.0, 200.0, – 1st point at (200,200)
  300.0, 300.0, – 2nd point at (300,300)
  200.0, 300.0); –3rd point at (200,300)
- Circle circle = new Circle();
- circle.setCenterX(100), circle.setCenterY(200); circle.setRadius(50);

- Image image = new Image("<image file name>") – images placed in src
- ImageView imageView = new ImageView(image)
- imageView.setX(500); imageView.setX(600)

JavaFX Buttons

Types

- Normal – Push button
- Default – default button that receives a keyboard VK_ENTER press
- Cancel – cancel button that receives a keyboard VK_ENTER press

Constructor – Button([String text], [Node icon])