Objective

Graphical Applications

- Creating Graphical Application
- Structure of Graphical Elements
- Basic Elements
  - Shapes, lines, text…

Graphical Applications

Java has included different packages for creating graphical programs.

- AWT
- Swing
- JavaFX

Swing

Swing came after AWT. The central idea behind Swing was not to use the native controls, but to paint its own.

That way, the user interface would look and feel the same on every platform.

Problems:

There were slow

Swing was ugly—indeed, it had fallen behind the native controls that had been spruced up with animations and fancy effects.

JavaFX

In 2007, Sun Microsystems introduced a new technology, called JavaFX, as competitor to Flash.
Java 8 was called JavaFX 8.
JavaFX versions 9 and 10 were bundled with Java 9 and 10.
From Java11, JavaFX is not bundles as part of standard Java library, and JavaFX is contiue as a separate open source project.

JavaFX

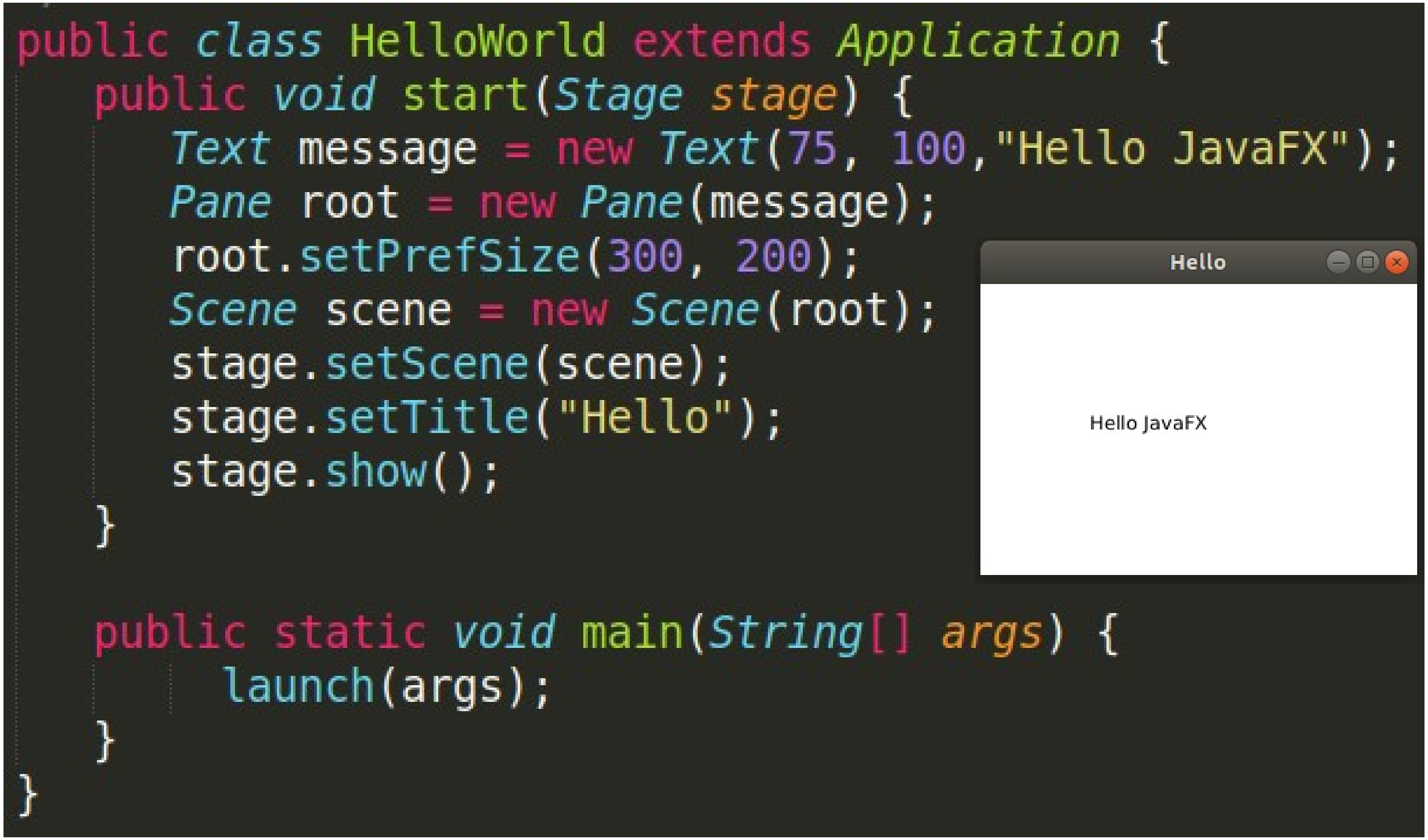
- We will focus on the Java aspect, but JavaFX has some other powerful tools
  - JavaFX allows CSS for styling
  - FXML a language to specify the graphical elements
- Nearly everything is represented by a class
  - Shapes
  - Colors
  - Fonts
  - Effects
  - ...

Java Version

IMPORTANT

- You must have the correct version of Java installed. We will be using **Java 8 (1.8)**.
  - Java 8 through 10 incorporate JavaFX into the standard release
- If you are using Eclipse or other IDE, then you should set up them properly to use JavaFX.

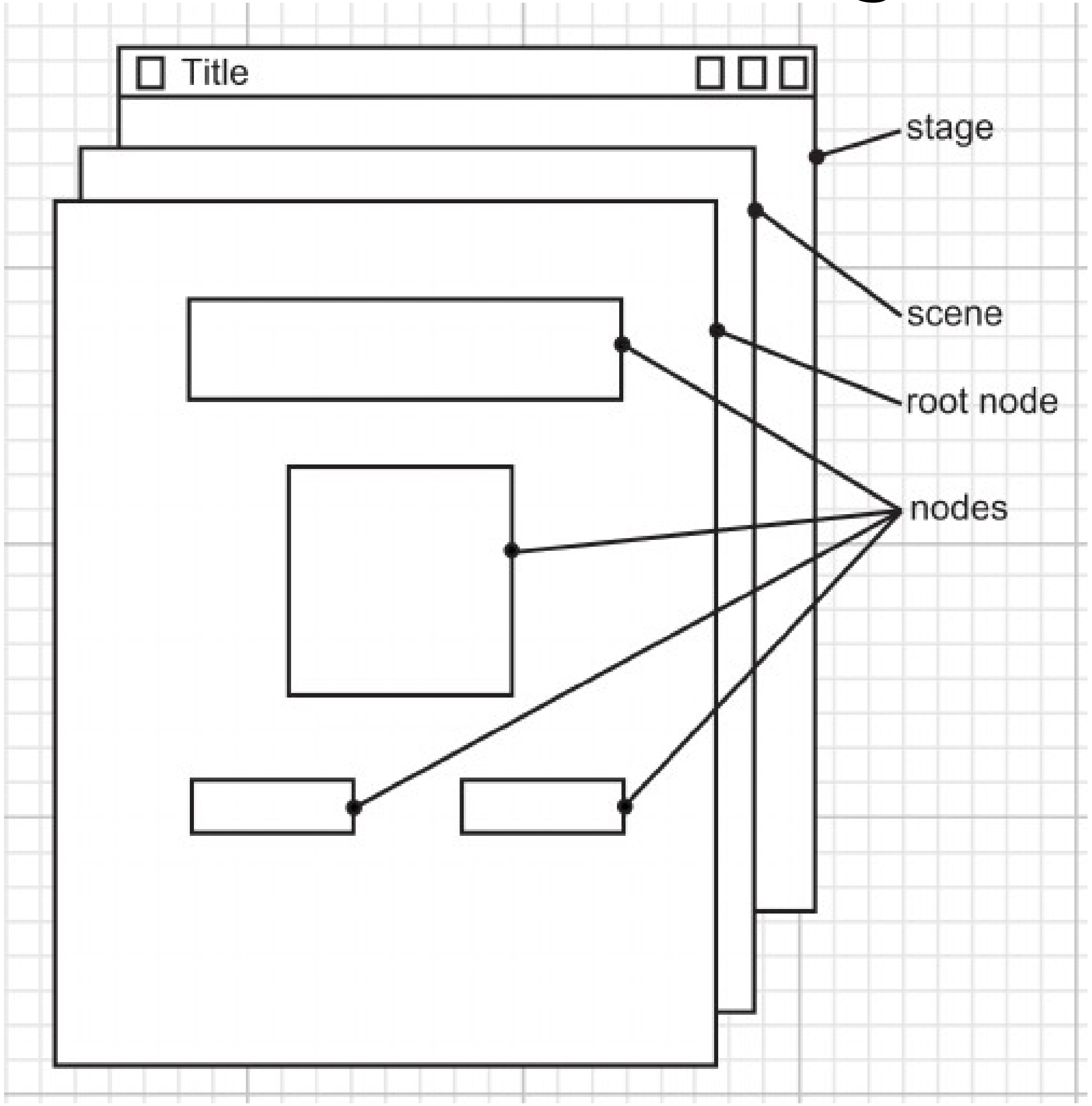
First JavaFX Application



```java
public class HelloWorld extends Application {
    public void start(Stage stage) {
        Text message = new Text(75, 100,"Hello JavaFX");
        Pane root = new Pane(message);
        root.setPrefSize(300, 200);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Hello");
        stage.show();
    }

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

Refer to HelloWorldFX.java

Structure of a Stage



JavaFX Application Structure

The structure of JavaFX graphic program.

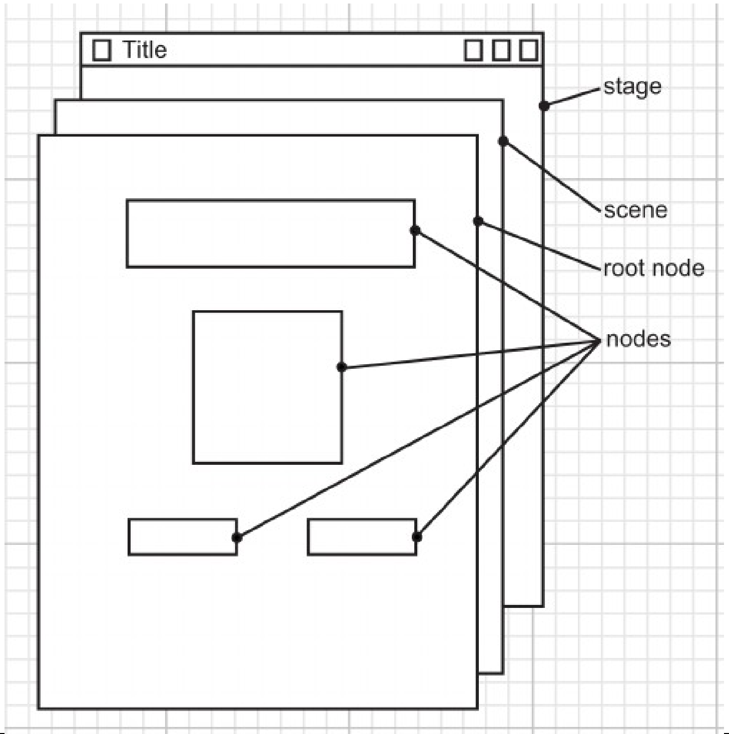
- Stage
  - the window
- Scene
  - the contents of a stage. One scene is displayed at a time
- Graphical Elements
  - Layout (root/Parent) elements that arrange elements inside them
    - Pane, Group, Region, Vbox, Hbox, BorderPane…
  - Display elements
    - Shapes, lines, text, textboxes, buttons…

Note: Anything that you display in JavaFX is a Node, like shape, text, button, user interface, ect.

JavaFX Application Structure

To display elements (some flexibility in order)

- Create a layout element, root
- Add display/layout elements to root
- Create a Scene object
- Add root to the Scene object
- Set the scene. (Add scene to the stage)



Example: Empty Window

```
public void start(Stage stage){
  // 1. create a root
  Pane root = new Pane();
  // 2. set the size of the root
  root.setPrefSize(400,200);
  // 3. create a Scene object
  Scene scene = new Scene(root);
  // 4. set the scene to the stage
  stage.setScene(scene);
  // 5. give a title (optional)
  stage.setTitle("Empty");
  // 6. show the window
  stage.show();
}
```

Refer to E1_EmptyFX.java

Example: Adding Text

```
// first create a Text
Text text1 = new Text(100,50,"first line");
//then add to the root at creation time
Pane root = new Pane(text1);

//or, create a Text
Text text2 = new Text(100,100,"second line");
//added to the root
root.getChildren().add(text2);
```

Refer to E2_TextFX.java

Example: Adding Text

// You can add as many as shapes or texts to the root at creation time

```
Pane root = new Pane(text1,text2, shape1,...);
```

// By invoking getChilrent() method of the root, we can access all elements in the root.

```
ObservableList  list = root.getChildren();
```

// returns a List of items in the root, and then we invoke its add method, to add a new item to the list.

```
list.add(text2);
```

//In the example TextFX.java, we developed a short version of the code:

```
root.getChildren().add(text2);
```

JavaFX Application

To create JavaFX a program, extend the Application class which contains

- `public void init() {}`
- **`public abstract void start(Stage primaryStage)`**
- `public void stop() {}`

Override start, and include code to build up the graphic elements

- Optionally override init/stop if you need code to run before start or when the application is closed.

In the main method call **`Application.launch(args);`**

Stage

Stage is the class that represents a window.

- setTitle(...) sets the text displayed in the title bar
- show() displays the window

```
public void start(Stage primaryStage) {
   primaryStage.setTitle("First JavaFX");
   primaryStage.show();
}
```

JavaFX Application

```
public void start(Stage primaryStage) {
   Text hello = new Text(50, 100, "Hello, World!");

   Pane root = new Pane();
   root.getChildren().add(hello);

   Scene scene = new Scene(root, 400, 200);
   primaryStage.setTitle("First JavaFX Example");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

JavaFX Application Structure

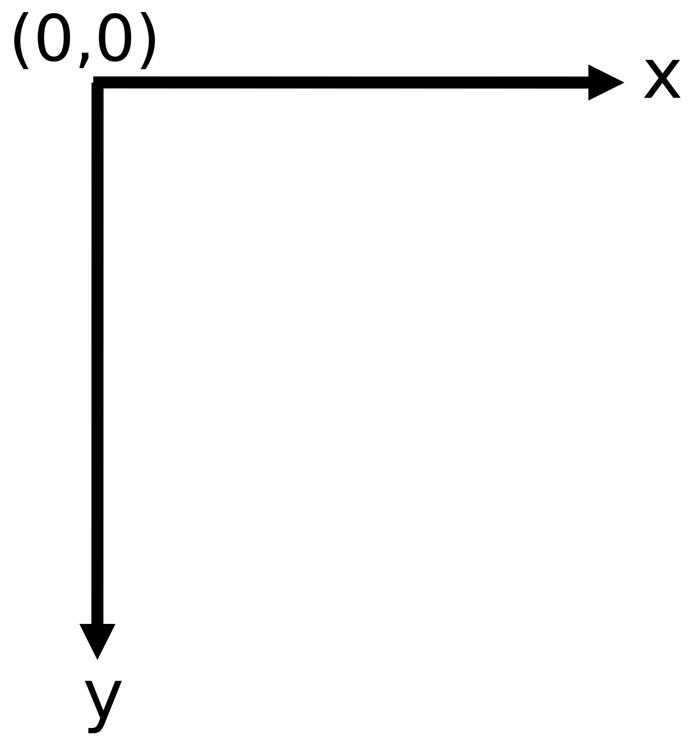
- The Pane object as well as other layout elements can contain other layout/display elements.
- .getChildren() gives access to the list of nodes, so you can add to it or modify it in other ways

```
Pane root = new Pane();
root.getChildren().add(hello);
```

- The elements in a Scene make up a tree structure that gives a clear way to visualize the relationship between elements.
  - Even though it is specifically a tree, it is called the **scene graph**.

Coordinate System

In computer Graphics The origin (0, 0) starts at the top left corner. The x-axis extends to the right and the y-axis extends to the bottom.



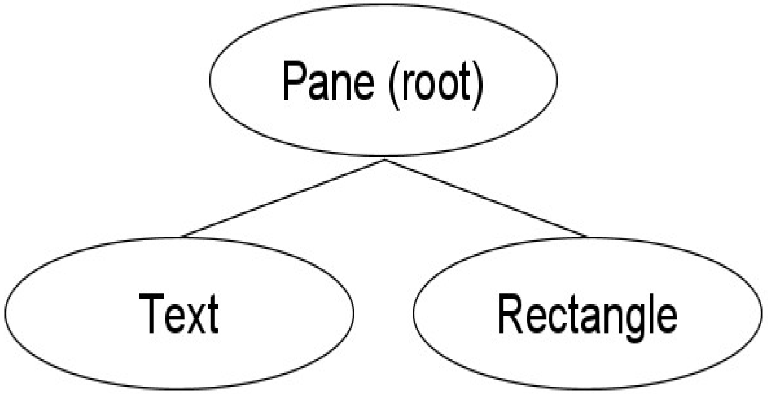
Adding a Rectangle

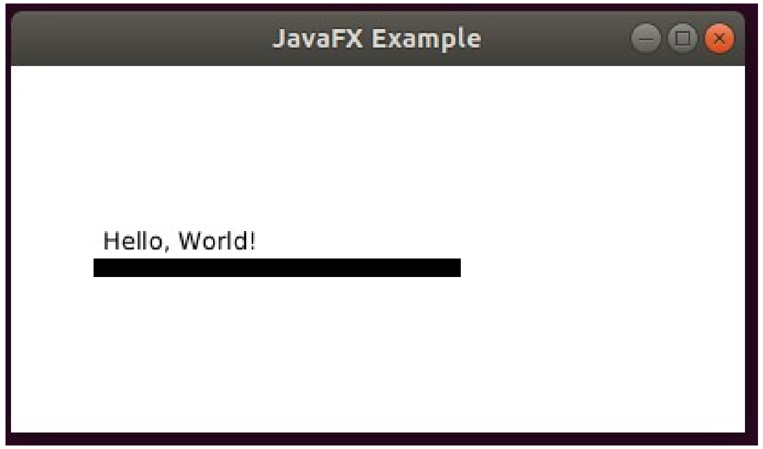
```
public void start(Stage primaryStage) {
   Text hello = new Text(50, 100, "Hello, World!");
   Rectangle r = new Rectangle(45, 105, 200, 10);
  //x,y,width,height
   Pane root = new Pane();
   root.getChildren().addAll(hello, r);
   Scene scene = new Scene(root, 400, 200);
   primaryStage.setTitle("JavaFX Example");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

Refer to E3_FirstShapeFX.java

Scene Graph

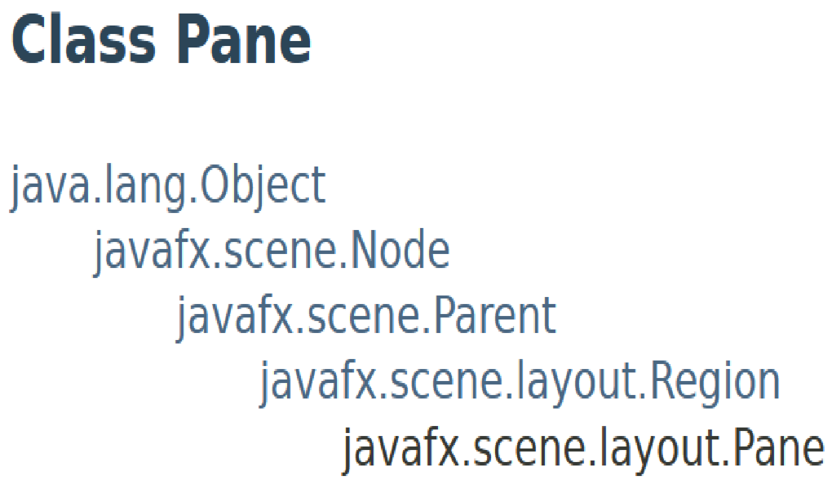
- Each element is a node in the tree.
- The root has no parents, all other nodes have one parent.
- Nodes that have children are parents.
- For now we will just add more children to the root element, but more complex layouts will lead to a more complex tree.





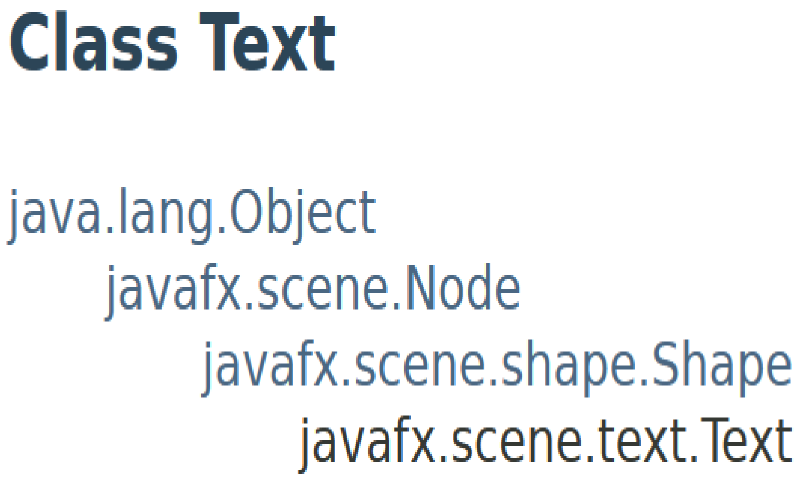
Scene Graph

- Node and Parent are actually classes in JavaFX
- Parent extends Node, so all Parents are Nodes
- If an element’s class extends Parent, it can contain children.



Class Pane

```
java.lang.Object
    javafx.scene.Node
        javafx.scene.Parent
            javafx.scene.layout.Region
                javafx.scene.layout.Pane
```



Class Text

```
java.lang.Object
    javafx.scene.Node
        javafx.scene.shape.Shape
            javafx.scene.text.Text
```

Rectangles

JavaFX Rectangle has a constructor takes x and y coordinates and its width and height.

Rectangle r = new Rectangle(x, y, w, h);

By default it will be drawn filled in as black.

To change the color use r.setFill(Paint p);

We will use the Paint class to set a color.

Paint is the abstract superclass of Color, ImagePattern, and two types of gradient.

There are constants for (almost) all named CSS colors :Color.RED, Color.YELLOW, Color.MINTCREAM, etc.

Full list: https://www.w3schools.com/colors/colors_names.asp

Colors

- On computers, we often use RGB values to choose a color.
- Color.rgb(int red, int green, int blue) lets you create any color you need
- Uses one byte to store each of the R, G and B values
  - 0-255

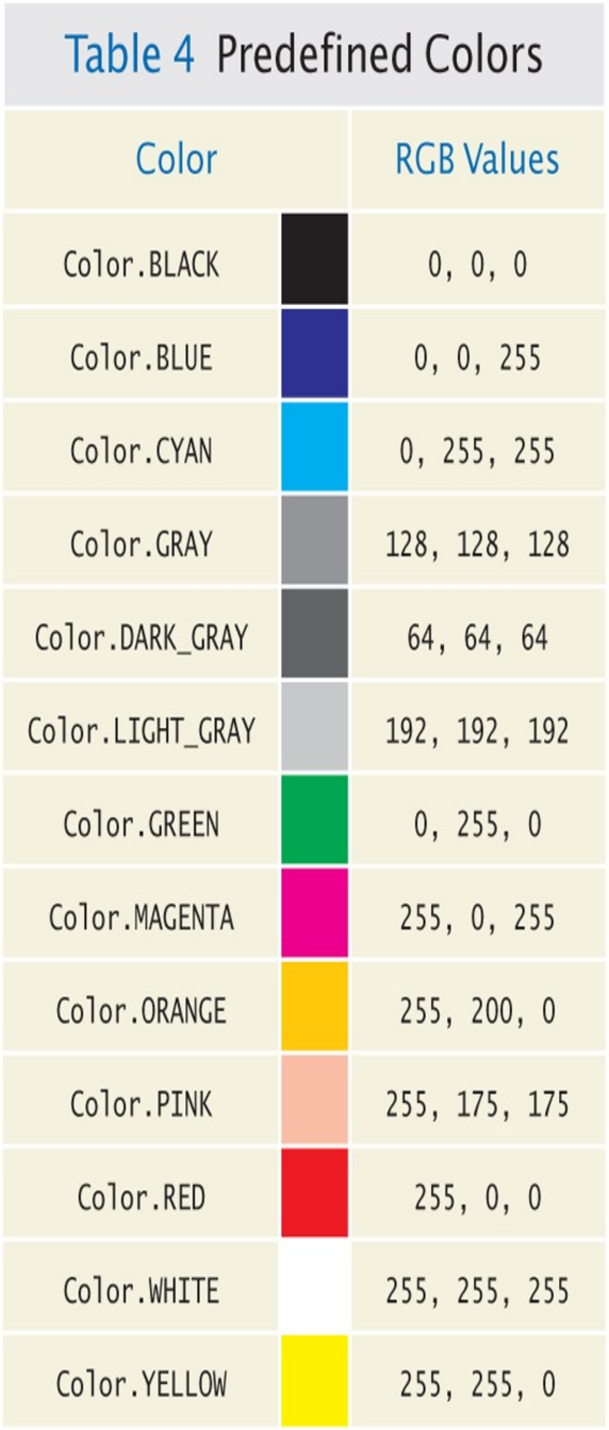


Table 4 Predefined Colors

Color		RGB Values
`Color.BLACK`	■	0, 0, 0
`Color.BLUE`	■	0, 0, 255
`Color.CYAN`	■	0, 255, 255
`Color.GRAY`	■	128, 128, 128
`Color.DARK_GRAY`	■	64, 64, 64
`Color.LIGHT_GRAY`	■	192, 192, 192
`Color.GREEN`	■	0, 255, 0
`Color.MAGENTA`	■	255, 0, 255
`Color.ORANGE`	■	255, 200, 0
`Color.PINK`	■	255, 175, 175
`Color.RED`	■	255, 0, 0
`Color.WHITE`	□	255, 255, 255
`Color.YELLOW`	■	255, 255, 0

Shapes

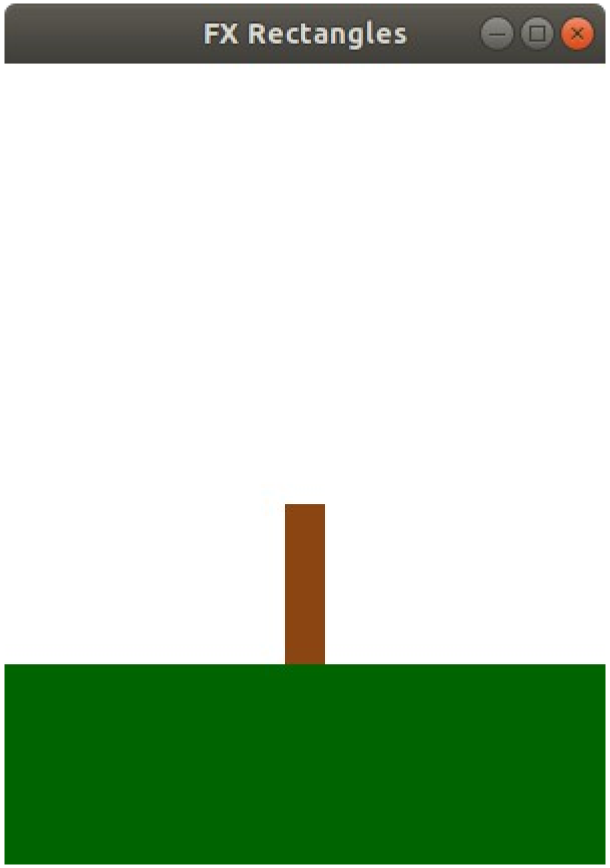
```
public void start(Stage primaryStage) {
   Pane root = new Pane();
   Rectangle ground = new Rectangle(0, 300, 300, 100);
   ground.setFill(Color.DARKGREEN);
   Rectangle trunk = new Rectangle(140, 220, 20, 100);
   trunk.setFill(Color.SADDLEBROWN);
   root.getChildren().addAll(ground, trunk);
   Scene scene = new Scene(root, 300, 400);
   primaryStage.setTitle(“FX Rectangles");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

```
java.lang.Object
   javafx.scene.paint.Paint
      javafx.scene.paint.Color
```

Refer to E4-RectanglesFX.java

Shapes

```
public void start(Stage primaryStage) {
   Pane root = new Pane();

   Rectangle ground = new Rectangle(0, 300, 300,
   ground.setFill(Color.DARKGREEN);
   Rectangle trunk = new Rectangle(140, 220, 20,
   trunk.setFill(Color.SADDLEBROWN);

   root.getChildren().addAll(ground, trunk);

   Scene scene = new Scene(root, 300, 400);
   primaryStage.setTitle(“FX Shapes");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```



addAll vs add

- You can add elements one at a time using add, or all at once with addAll
- The following two options have the same result

```
root.getChildren().addAll(r1,r2);



root.getChildren().add(r1);
root.getChildren().add(r2);
```

Ellipse

To create a rectangle, we need the top-left coordinate of the rectangle, and then width and height.

To create ellipse, we need the center of the shape and then horizontal and vertical radius of the ellipse.

```
Ellipse e = new
          Ellipse(centerx, centery, xRadius, yRadius);
```

Note: a Circle is an Ellipse with equal horizontal and vertical radius.

The fill color is set the same was as for Rectangle

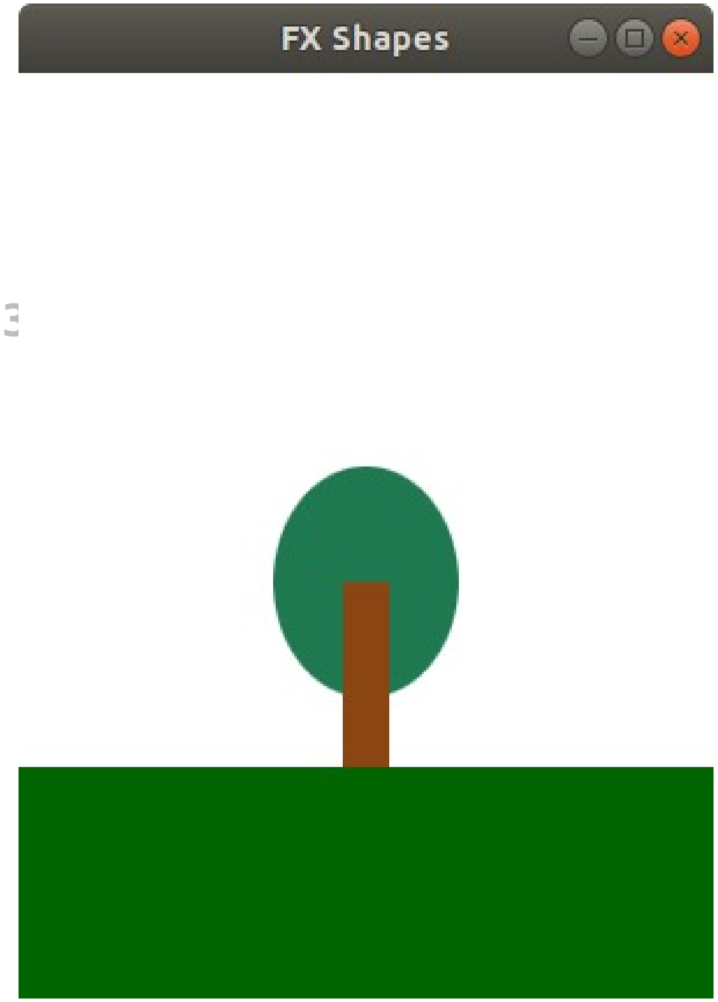
Ellipses

```
public void start(Stage primaryStage) {
   Pane root = new Pane();
   // Code to create ground/trunk rectangles removed
  // for clarification purpose

   Ellipse leaves = new Ellipse(150,220,40,50);

   leaves.setFill(Color.rgb(30,120,80));

   root.getChildren().addAll(ground, trunk, leaves);
   Scene scene = new Scene(root, 300, 400);
   primaryStage.setTitle(“FX Shapes");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

Refer to E5_ShapesFX.java

Shapes

```
public void start(Stage primaryStage) {
   Pane root = new Pane();

   Rectangle ground = new Rectangle(0, 300, 3
   ground.setFill(Color.DARKGREEN);
   Rectangle trunk = new Rectangle(140, 220,
   trunk.setFill(Color.SADDLEBROWN);

   root.getChildren().addAll(ground, trunk);

   Scene scene = new Scene(root, 300, 400);
   primaryStage.setTitle(“FX Shapes");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

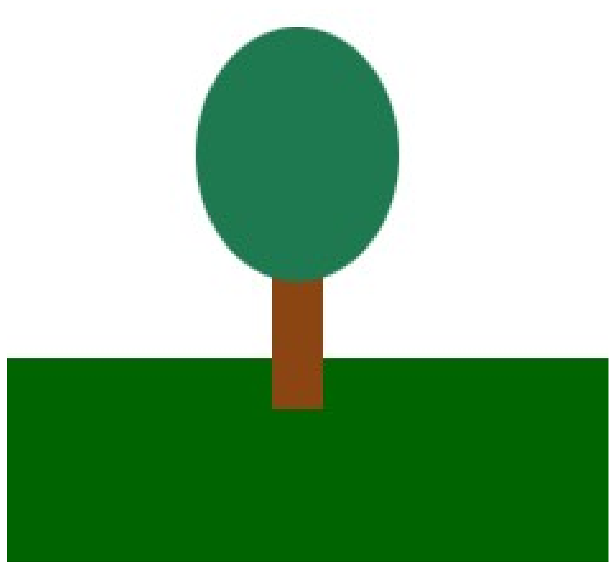


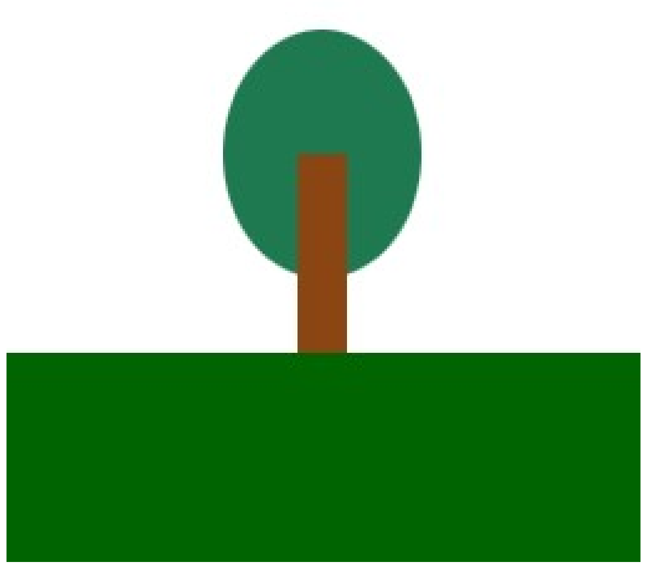
Overlapping Elements

When multiple elements overlap, each element is drawn covering up previously added elements

Left: `root.getChildren().addAll(`**`ground, trunk, leaves`**`);`

Right: `root.getChildren().addAll(`**`leaves, trunk, ground`**`);`





Adding an Outlines (Borders)

Outlines can be added using:

setStroke(Paint p)

The width is adjusted with:

setStrokeWidth(int width)

If you only want an outline, set the fill to:

Color.TRANSPARENT

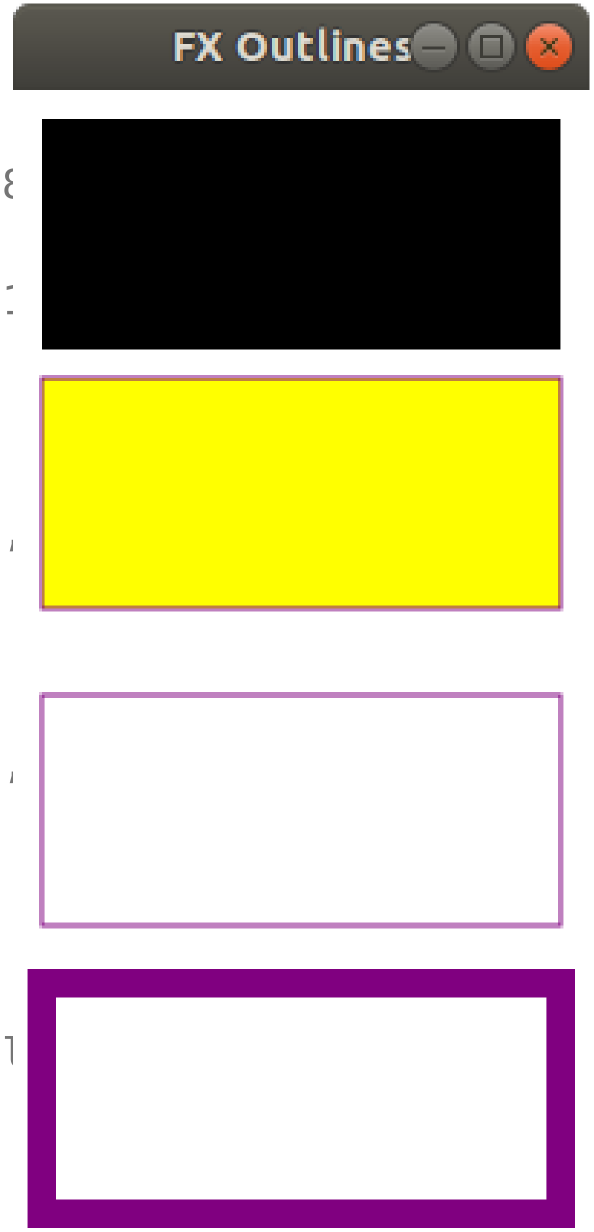
Adding Outline

```
public void start(Stage primaryStage) {
  Pane root = new Pane();
  Rectangle defaultRect = new Rectangle(10, 10, 180, 80);

  Rectangle lineAndFill = new Rectangle(10, 100, 180, 80);
  lineAndFill.setFill(Color.YELLOW);
  lineAndFill.setStroke(Color.PURPLE);

  Rectangle lineOnly = new Rectangle(10, 210, 180, 80);
  lineOnly.setFill(Color.TRANSPARENT);
  lineOnly.setStroke(Color.PURPLE);

  Rectangle wideLine = new Rectangle(10, 310, 180, 80);
  wideLine.setFill(Color.TRANSPARENT);
  wideLine.setStroke(Color.PURPLE);
  wideLine.setStrokeWidth(10);

  root.getChildren().addAll(defaultRect,lineAndFill, lineOnly, wideLine);
  Scene scene = new Scene(root, 200, 400);
   //code to setup stage removed
}
```

Refer to E6_OutlinesFX.java

Adding Outline

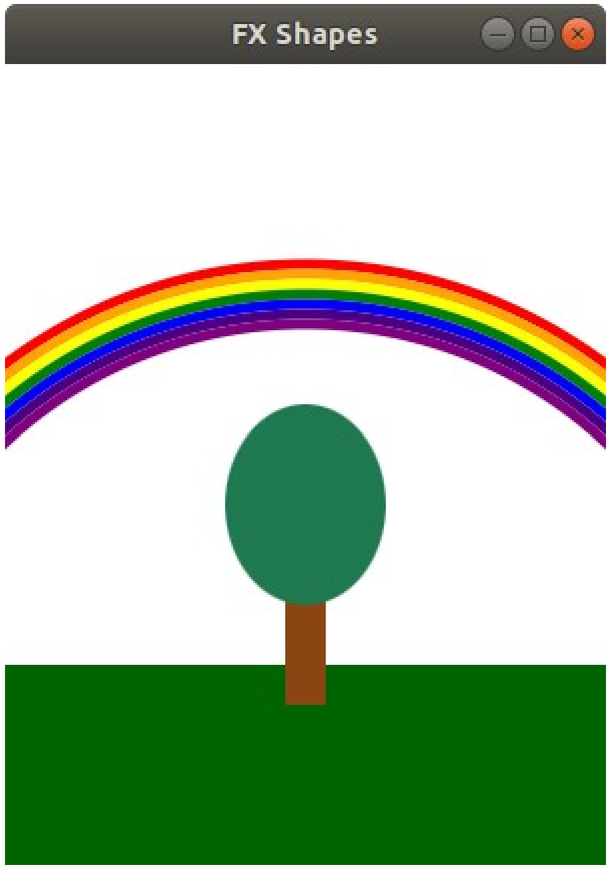
```
public void start(Stage primaryStage) {
  Pane root = new Pane();
  Rectangle defaultRect = new Rectangle(10, 10, 18

  Rectangle lineAndFill = new Rectangle(10, 100, 1
  lineAndFill.setFill(Color.YELLOW);
  lineAndFill.setStroke(Color.PURPLE);

  Rectangle lineOnly = new Rectangle(10, 210, 180,
  lineOnly.setFill(Color.TRANSPARENT);
  lineOnly.setStroke(Color.PURPLE);

  Rectangle wideLine = new Rectangle(10, 310, 180,
  wideLine.setFill(Color.TRANSPARENT);
  wideLine.setStroke(Color.PURPLE);
  wideLine.setStrokeWidth(10);

  root.getChildren().addAll(defaultRect,lineAndFil
  Scene scene = new Scene(root, 200, 400);
   //code to setup stage removed
}
```



Exercise: Adding a Rainbow

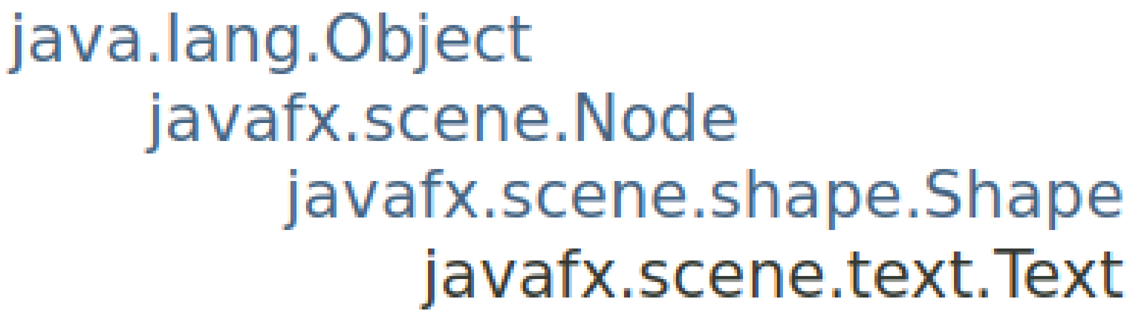
- Add a rainbow arcing over the tree.
- Using the tools shown so far, how would you draw the rainbow?
- Starting with the provided code (E5_ShapesFX.java) , try it yourself.
  - Create and set the properties of the needed shapes and add them to the root element
  - Try creating 2 or 3 stripes
  - If successful, can you find a way to use a loop and or a function to avoid repeating too much code?

  Note: use Ellipse to create the rainbow



Text

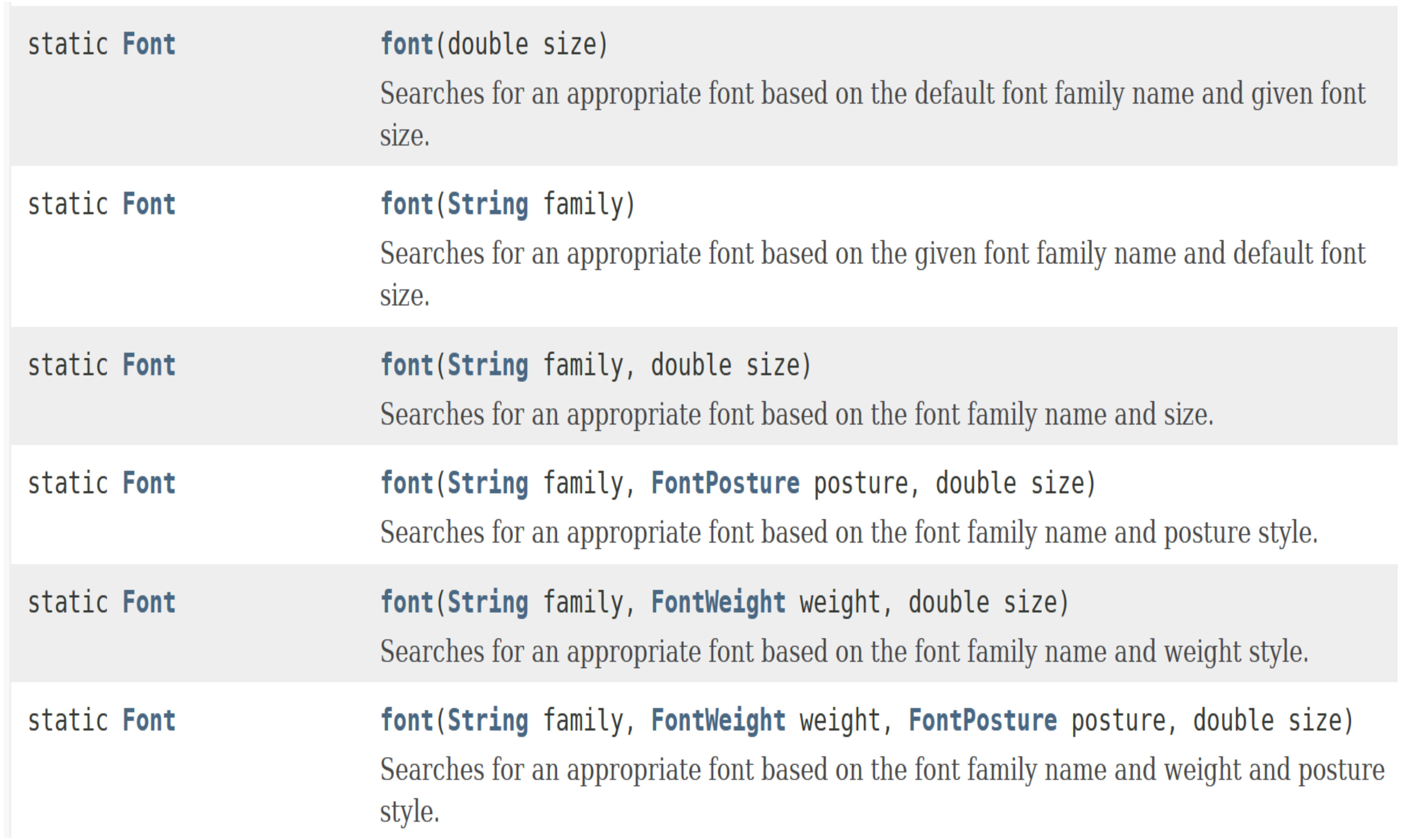
- The Text class is used to add text.
- One constructor sets the x,y coordinates and the text to display
  - `Text t = new Text(double x, double y, String message");`
- It works the same as Rectangle and Ellipse for setting fill and stroke.
  - because it also extends the Shape class
- Use setFont(Font f) to change font, size, bold, italic, etc.



```
java.lang.Object
    javafx.scene.Node
        javafx.scene.shape.Shape
            javafx.scene.text.Text
```

Font

The JavaFX Font class has two Constructors, but has more flexible static methods for creating a Font object.

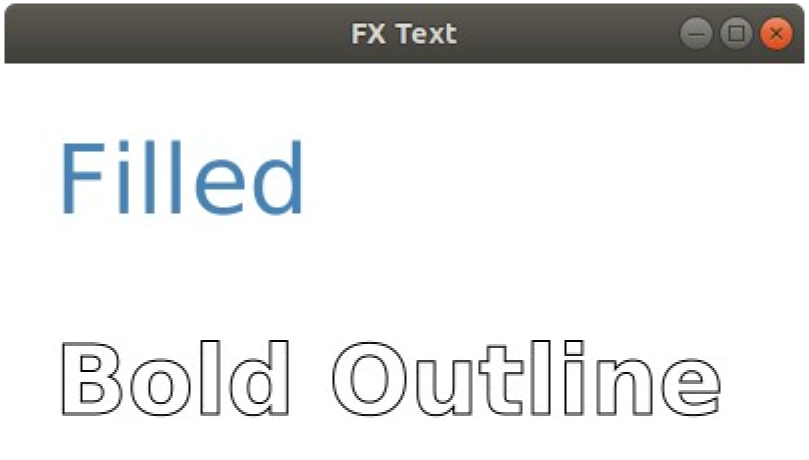


static Font	font(double size) Searches for an appropriate font based on the default font family name and given font size.
static Font	font(String family) Searches for an appropriate font based on the given font family name and default font size.
static Font	font(String family, double size) Searches for an appropriate font based on the font family name and size.
static Font	font(String family, FontPosture posture, double size) Searches for an appropriate font based on the font family name and posture style.
static Font	font(String family, FontWeight weight, double size) Searches for an appropriate font based on the font family name and weight style.
static Font	font(String family, FontWeight weight, FontPosture posture, double size) Searches for an appropriate font based on the font family name and weight and posture style.

Text/Font

```
public void start(Stage primaryStage) {
  Pane root = new Pane();

  Text t1 = new Text(25, 75, "Filled");
  t1.setFont(Font.font(48));
  t1.setFill(Color.STEELBLUE);

  Text t2 = new Text(25, 175, "Bold Outline");
  Font f = Font.font("Times New Roman", FontWeight.BOLD ,48);
  t2.setFont(f);
  t2.setFill(Color.TRANSPARENT);
  t2.setStroke(Color.BLACK);

  root.getChildren().addAll(t1, t2);

  Scene scene = new Scene(root, 400, 200);
  primaryStage.setTitle("FX Text");
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

Font



```
public void start(Stage primaryStage) {
   Pane root = new Pane();

   Text t1 = new Text(25, 75, "Filled");
   t1.setFont(Font.font(48));
   t1.setFill(Color.STEELBLUE);

   Text t2 = new Text(25, 175, "Bold Outline");
   Font f = Font.font("Times New Roman", FontWeight.BOLD ,48);
   t2.setFont(f);
   t2.setFill(Color.TRANSPARENT);
   t2.setStroke(Color.BLACK);

   root.getChildren().addAll(t1, t2);

   Scene scene = new Scene(root, 400, 200);
   primaryStage.setTitle("FX Text");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

Refer to E7_FontFX.java

Building Shape Classes

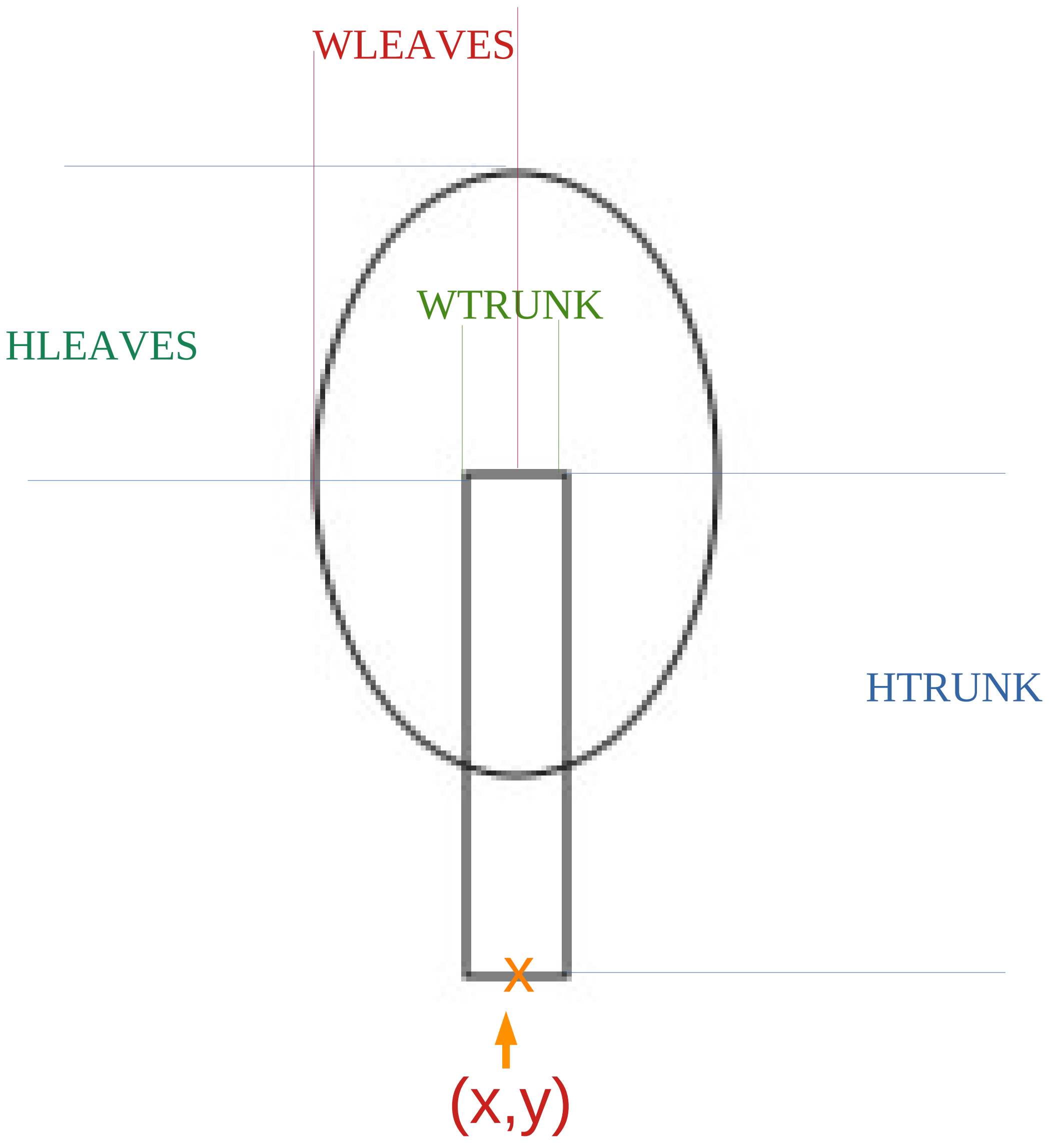
- We made a tree earlier
- What if we want to make many trees?
- Let's create a class for Tree
  - getAllNodes will return all the elements that need to be added

```
public class Tree{
  public Tree (int x, int y){}
  public Node [] getAllNodes(){}
}
```

Drawing the Tree

- Turning the code for drawing a tree into a generic method for drawing the tree is the hardest part
- The tree will be positioned at point (x,y)
- The very first thing we need to decide is where that point is on the tree
- I chose it to represent the bottom center of the trunk
- Now we need to rewrite all of the positioning commands to be relative to this x,y point
    - Sketching it out can help

Schetch



Tree

```
class Tree {
  private Rectangle trunk;
  private Ellipse leaves;
  private final int WTRUNK = 20;
  private final int HTRUNK = 100;
  private final int WLEAVES = 40;
  private final int HLEAVES = 60;
  public Tree(int x, int y) {
    trunk = new Rectangle(x-WTRUNK/2, y-HTRUNK, WTRUNK, HTRUNK);
    trunk.setFill(Color.SADDLEBROWN);
    leaves = new Ellipse(x,y-HTRUNK,WLEAVES,HLEAVES);
    leaves.setFill(Color.rgb(30,120,80));
  }
  public Node[] getAllNodes() {
    return new Node[] {trunk, leaves};
  }
}
```

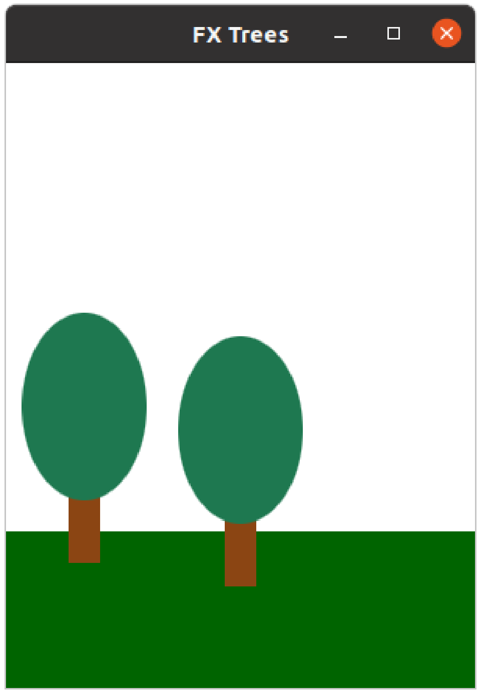
Adding Trees

```
public void start(Stage primaryStage) {
  Pane root = new Pane();
  Rectangle ground = new Rectangle(0, 300, 300, 100);
  ground.setFill(Color.DARKGREEN);
  root.getChildren().add(ground);

  Tree t1 = new Tree(50, 320);
  root.getChildren().addAll(t1.getAllNodes());

  Tree t2 = new Tree(150, 335);
  root.getChildren().addAll(t2.getAllNodes());

  Scene scene = new Scene(root, 300, 400);
  primaryStage.setTitle("FX Trees");
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

Refer to E8_TreesFX.java

Adding Trees

```
public void start(Stage primaryStage) {
   Pane root = new Pane();
   Rectangle ground = new Rectangle(0, 300, 300, 100);
   ground.setFill(Color.DARKGREEN);
   root.getChildren().add(ground);

   Tree t1 = new Tree(50, 320);
   root.getChildren().addAll(t1.getAllNodes());

   Tree t2 = new Tree(150, 335);
   root.getChildren().addAll(t2.getAllNodes());

   Scene scene = new Scene(root, 300, 400);
   primaryStage.setTitle("FX Trees");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```



Refer to TreesFX.java

Encapsulation

Does anything about getAllNodes() seem a bit…strange?

```
public Node[ ] getAllNodes() {
    return new Node[ ] {trunk, leaves};
}
```

It is making a copy of the private data members. It isn't maintaining encapsulation.

While, this may be ok, let's avoid it to emphasize encapsulation (as it is a central theme of our course).

Encapsulation

Is there a place we could write the Tree class and give it easy access to the root Pane?

Let’s make Tree an **inner class.**

It can only be used in our one application, but that is often the way code is written

It can then access the private data members/methods of the outer Application class

Inner Class

```
public class ComplexShapesFX extends Application {
  private Pane root;

  @Override
  public void start(Stage primaryStage) {
  root = new Pane();

  Rectangle ground = new Rectangle(0, 300, 300, 100);
  ground.setFill(Color.DARKGREEN);

  root.getChildren().addAll(ground);

  Tree t1 = new Tree(50, 320);

  Tree t2 = new Tree(150, 340);

  Scene scene = new Scene(root, 300, 400);
  primaryStage.setTitle("FX Trees");
  primaryStage.setScene(scene);
  primaryStage.show();
  }
```

Inner Class

```
public class TreesInnerFX extends Application {
   private Pane root;
   @Override
   public void start(Stage primaryStage) {…   }

   private class Tree {
    ….
    public Tree(int x, int y) {
    trunk = new Rectangle(x-WTRUNK/2, y-HTRUNK, WTRUNK, HTRUNK);
    trunk.setFill(Color.SADDLEBROWN);

    leaves = new Ellipse(x,y-HTRUNK,WLEAVES,HLEAVES);
    leaves.setFill(Color.rgb(30,120,80));
    // add directly to root
    root.getChildren().addAll(trunk,leaves);
  }
}
```

Refer to TreesInnerFX.java

Exercise

How would you modify the Tree class to be able to draw Trees of varying sizes?

Lines

Line has a constructor that takes x and y coordinates for its endpoints.

```
Line l = new Line(startX, startY, endX, endY);
```

Since it is a line, there is not setfill() method.

Use setStrokeWidth() and setStroke() to customize the line

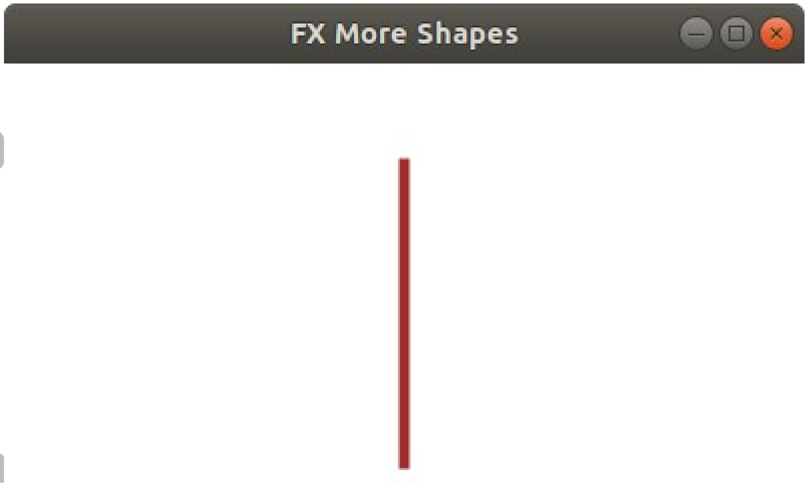
Line

```
@Override
public void start(Stage primaryStage) {
   Pane root = new Pane();

   Line mast = new Line(200, 50, 200, 200);
   mast.setStrokeWidth(5);
   mast.setStroke(Color.BROWN);

   root.getChildren().addAll(mast);

   Scene scene = new Scene(root, 400, 300);
   primaryStage.setTitle("FX More Shapes");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

Refer to E10_LineFX.java

Line

```
@Override
public void start(Stage primaryStage) {
   Pane root = new Pane();

   Line mast = new Line(200, 50
   mast.setStrokeWidth(5);
   mast.setStroke(Color.BROWN);

   root.getChildren().addAll(ma

   Scene scene = new Scene(root
   primaryStage.setTitle("FX More Shapes");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```



Polygon

Polygon lets you create your own shape with as many sides as you wish.

Give the constructor a series of points specified by an array of doubles

`[x1,y1, x2,y2, …]`

The last point will automatically connect to the first.

```
 Polygon p = new Polygon(double... points);
```

You can set the fill and line properties with the functions previously seen

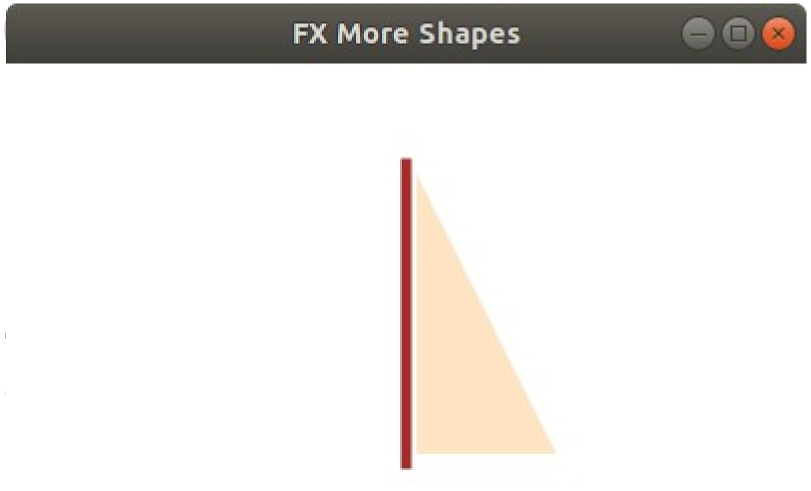
Polygon

```
@Override
public void start(Stage primaryStage) {
   Line mast = new Line(200, 50, 200, 200);
   mast.setStrokeWidth(5);
   mast.setStroke(Color.BROWN);

   double[] sailPoints = {205,55 , 205,195 , 275,195};
   Polygon sail = new Polygon(sailPoints);
   sail.setFill(Color.BISQUE);

   root.getChildren().addAll(mast, sail);
```

Refer to E11_PolygonFX.java

Polygon

```
@Override
public void start(Stage primaryStage) {
    Line mast = new Line(200, 5
    mast.setStrokeWidth(5);
    mast.setStroke(Color.BROWN)

    double[] sailPoints = {205,
    Polygon sail = new Polygon(
    sail.setFill(Color.BISQUE);

    root.getChildren().addAll(m
```



Arc

Arc lets you have a portion of an ellipse.

Arc a=new Arc(centerX,centerY,radiusX,radiusY,

startAngle,angleLength)

The first four set up an ellipse as we've seen before.

startAngle and angleLength specify the portion of the ellipse in degrees

setType(ArcType a)

controls how the arc is completed

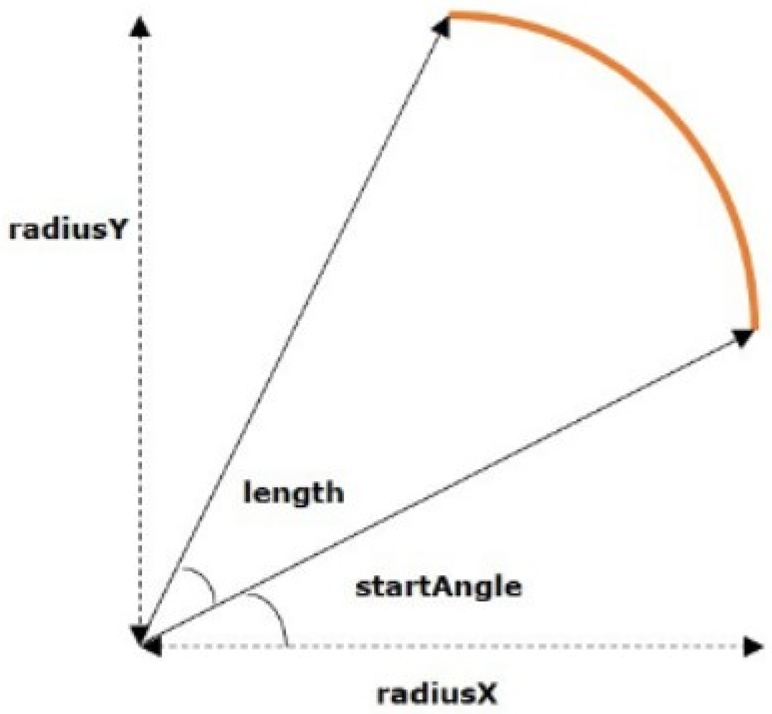
OPEN, ROUND, CHORD

Arc

What These parameter mean:

startAngle is the starting point (0 at rightmost point, goes counterclockwise)

angleLength (length) is how much of the ellipse to include



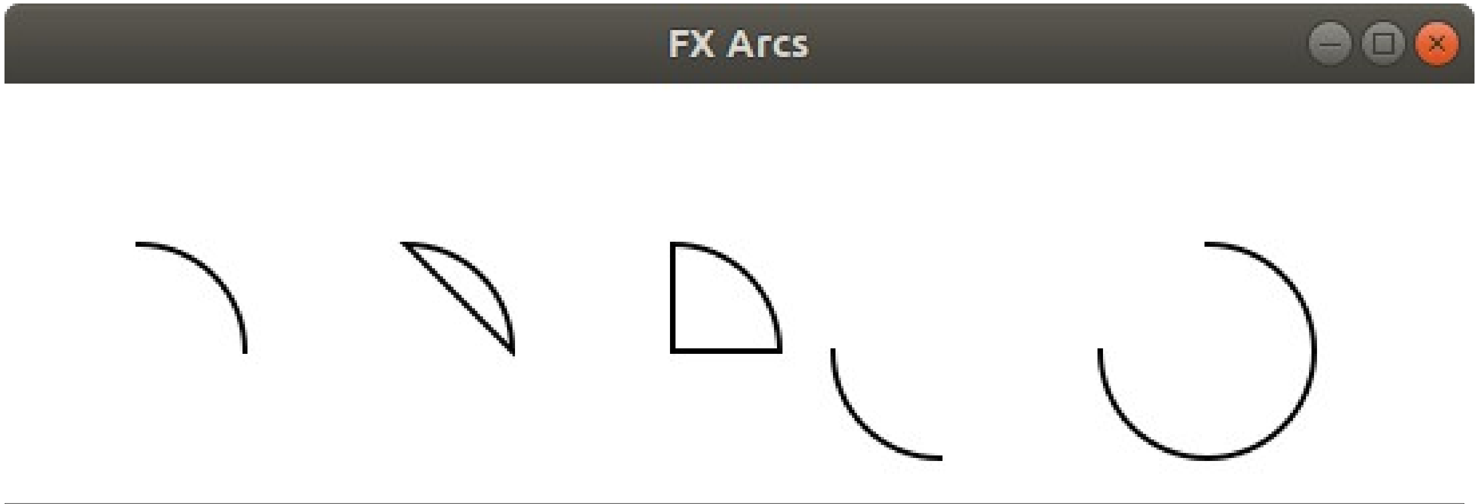
Arc

```
@Override
public void start(Stage primaryStage) {
  Arc a1 = new Arc(50, 100, 40, 40, 0, 90);
  Arc a2 = new Arc(150, 100, 40, 40, 0, 90);
  a2.setType(ArcType.CHORD);
  Arc a3 = new Arc(250, 100, 40, 40, 0, 90);
  a3.setType(ArcType.ROUND);
  Arc a4 = new Arc(350, 100, 40, 40, 180, 90);
  Arc a5 = new Arc(450, 100, 40, 40, 180, 270);
  Arc[] arcs = {a1,a2,a3,a4,a5};
  for (Arc a: arcs) {
    a.setFill(Color.TRANSPARENT);
    a.setStrokeWidth(2);
    a.setStroke(Color.BLACK);
  }
  root.getChildren().addAll(arcs);
```

Refer to SampleArcsFX.java

Arc

```
@Override
public
   Arc
   Arc
   a2.s
   Arc
   a3.s
   Arc
   Arc
   Arc[
   for
      a.
      a.setStrokeWidth(2);
      a.setStroke(Color.BLACK);
   }
   root.getChildren().addAll(arcs);
```



import javafx.scene.shape.ArcType;

From Left to right, ArcType:

OPEN, CORD, ROUND, OPEN, OPEN

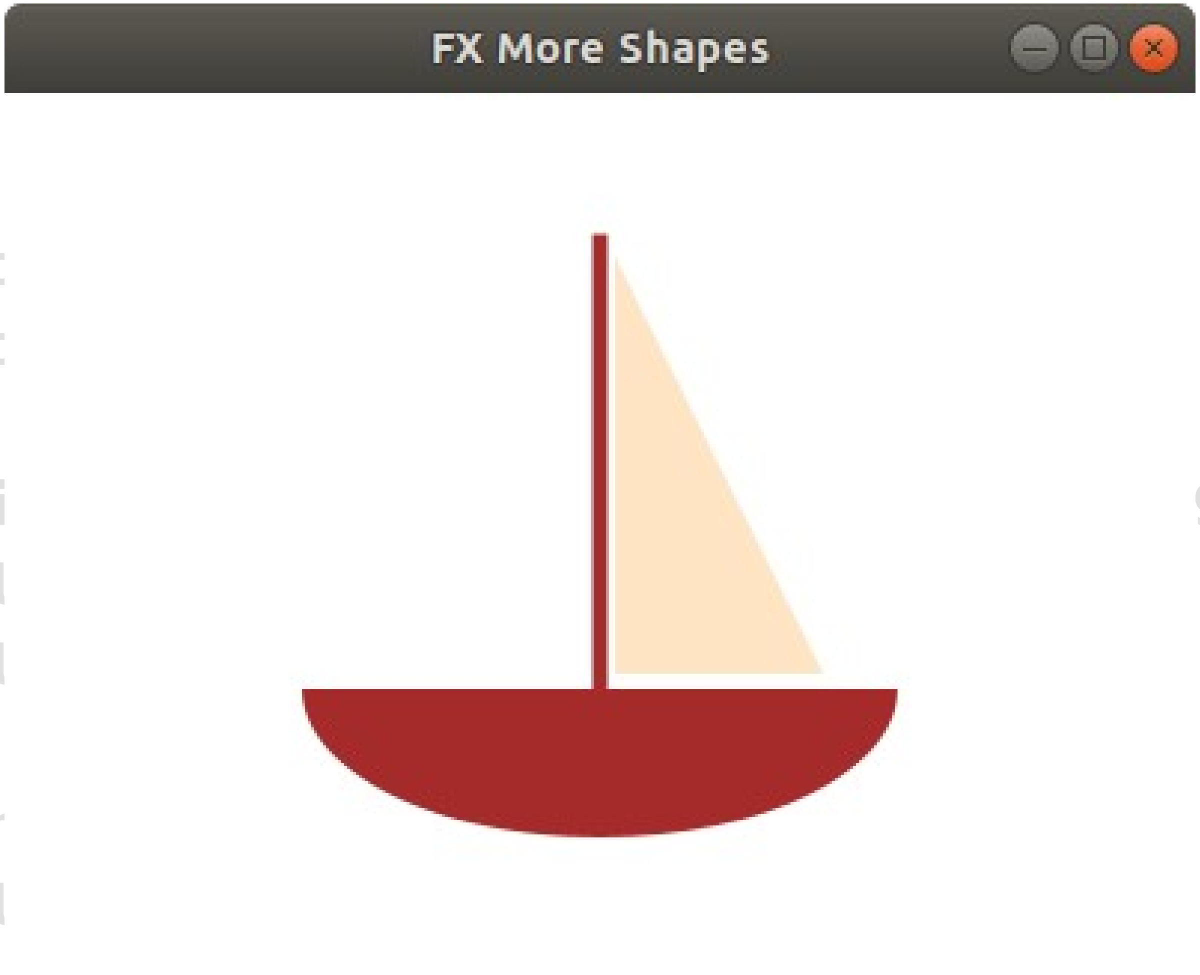
Arc

```
@Override
public void start(Stage primaryStage) {
   Line mast = new Line(200, 50, 200, 200);
   mast.setStrokeWidth(5);
   mast.setStroke(Color.BROWN);

   double[] sailPoints = {205,55 , 205,195 , 275,195};
   Polygon sail = new Polygon(sailPoints);
   sail.setFill(Color.BISQUE);
      Arc hull = new Arc(200,200,100,50,180,180);
   hull.setFill(Color.BROWN);
         root.getChildren().addAll(mast, sail, hull);
```

Refer to E13_BoatFX.java

Arc

```
@Override
public void st
   Line mast =
   mast.setStr
   mast.setStr

   double[] sai                                 95};
   Polygon sail
   sail.setFill

   Arc hull = r
   hull.setFill


   root.getChildren().addAll(mast, sail, hull);
```



Variable Argumen

```
public void fun(int ... a){
}
```

This syntax tells the compiler that fun( ) can be called with zero or more arguments. As a result, here a is implicitly declared as an array of type int[].

Refer to:

https://www.geeksforgeeks.org/variable-arguments-varargs-in-java/

Example

```
class VarArgs{
     public static void main(String args[])        {
       // Calling the varargs method with different number
       // of parameters
       fun(100);         // one parameter
       fun(1, 2, 3, 4);  // four parameters
       fun();            // no parameter
   }
   // A method that takes variable number of integer
   // arguments.
   static void fun(int ...a)  {
       System.out.println("Number of arguments: " + a.length);

       // using for each loop to display contents of a
       for (int i: a)
           System.out.print(i + " ");
       System.out.println();
   }
 }
```

Refer to VarArgs.java