# JavaFX: Basics

*First, go to Lec2\_Classe&Method, bring up the power point slides for Chpt3: Using Classes and Objects. Start from slide#67: Intro to JavaFX. Read all the slides from there to end and go over the examples. Then come to this lecture note.*

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 as opposed to web applications developed using AWT and Swing which are most appropriate for desktop applications. Written as a Java API, JavaFX application code can reference APIs from any Java library. JavaFX combines the best aspects of the previous approaches and adds many additional features.

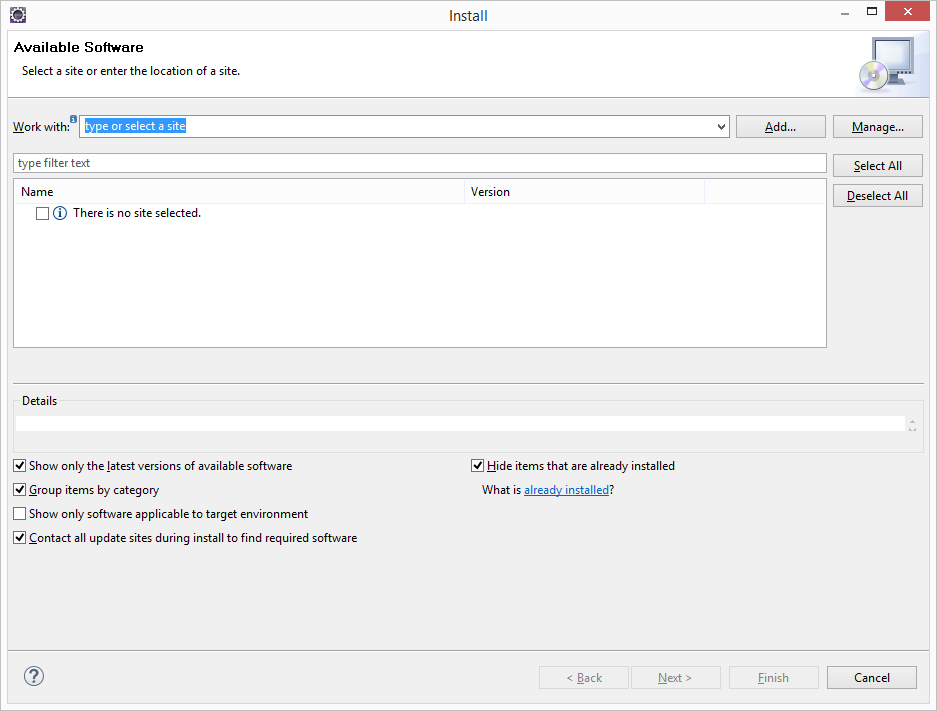
JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future. It was first made available on JDK7 and with the release of JDK 11 in 2018, Oracle has made JavaFX part of the OpenJDK under the OpenJFX project. JavaFX is also available, for the current long-term version (Java JDK 8), through March 2025.

# JavaFX for Eclipse:

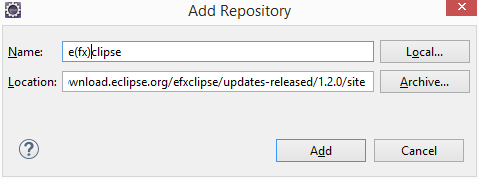
To enable eclipse to handle JavaFX programs, we will install a new software from eclipse.

You will need at least Java 8.

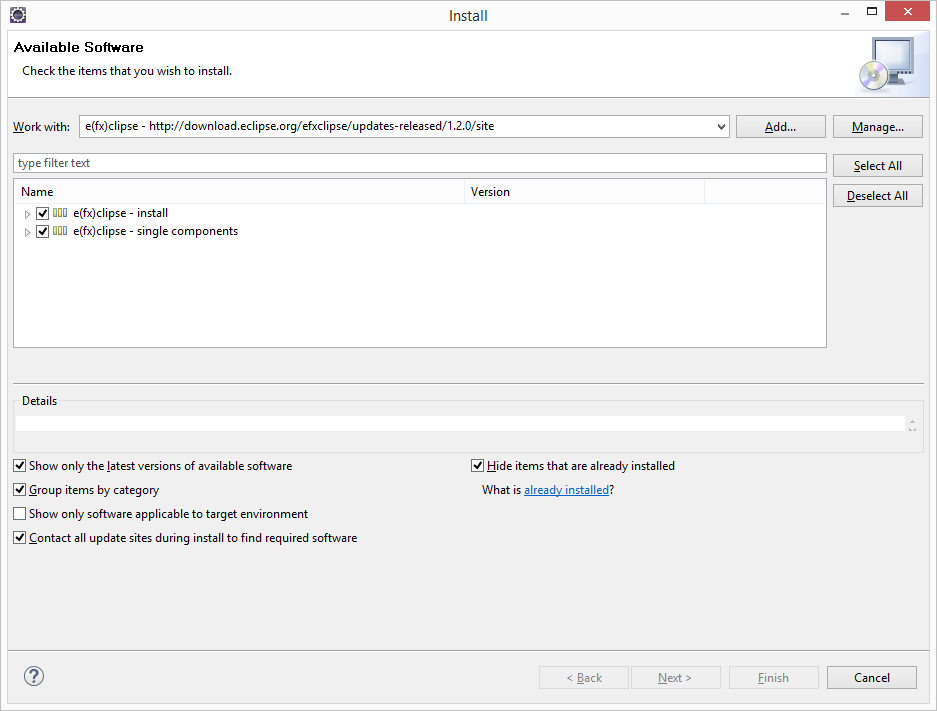
1. Open Eclipse and click on **Help**. Choose **Install a New Software** from the list of options given in the drop down menu.



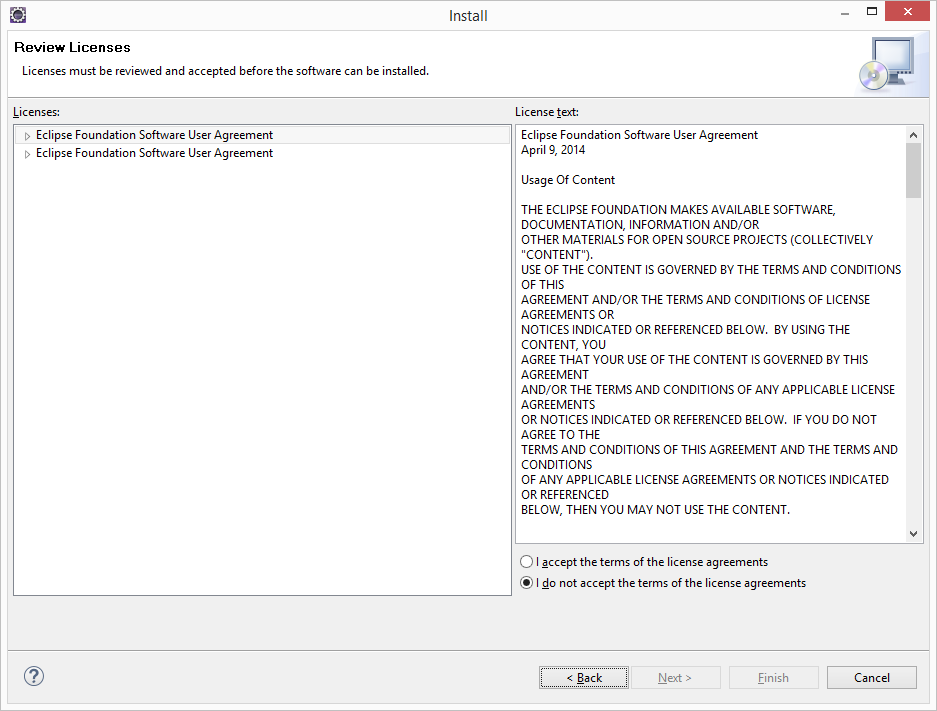
1. Now, we need to add a repository, just click on **Add. T**ype **e(fx)clipse** in the Name and type [http://download.eclipse.org/efxclipse/updates-released/1.2.0/site](http://download.eclipse.org/efxclipse/updates-released/1.2.0/site%20) in the location. Press OK to proceed with this repository installation.



1. Click on **Add**.
2. You might see the following window, check the two items and click **Next**.



1. A list with names of all the components which will be installed will be shown. Just click **Next** to continue with the installation.
2. Now, a software agreement has been shown. Just select the radio box which says **I accept the terms and conditions** and click **finish**.



1. Eclipse will ask you to restart. Please restart.

NB: Depending on Eclipse version you have installed, you might see different windows but in general the process will be similar.

# Example 1:

We write our first JavaFX program, HelloJavaFX.java, to understand the process and some key aspects of a JavaFX applications.

**import** javafx.application.Application;

**import** javafx.scene.Group;

**import** javafx.scene.Scene;

**import** javafx.scene.paint.Color;

**import** javafx.scene.text.Text;

**import** javafx.stage.Stage;

**public** **class** HelloFX **extends** Application {

//Creates and displays two Text objects in Javafx window

**public** **void** start(Stage primaryStage) {

Text hello = **new** Text (50, 50, "Hello, JavaFX!");

Text question = **new** Text(120, 80, "How's it going?");

Group root = **new** Group(hello, question);

Scene scene = **new** Scene(root, 300, 120, Color.***LIGHTGREEN***);

primaryStage.setTitle("A JavaFX Program");

primaryStage.setScene(scene);

primaryStage.show();

}

//Launches javafx application. this method is not required in IDEs

//that launch javafx applications automatically

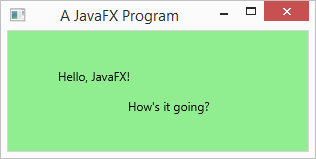
**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

When we run the program, the result shows:



## Description:

The first thing to note is that the HelloJavaFX class extends the JavaFX *Application* class.

**public** **class** HelloFX **extends** Application

All JavaFX programs extend the *Application* class.

**public** **void** start(Stage primaryStage)

There are two methods – the main() and the start(). The main method calls the launch() method of *Application* class automatically. It does some background work and then calls the start() method. Usually we will use the start() method to setup and display the primary window of the app. It takes an object of the *Stage* class as a parameter.

Text hello = **new** Text (50, 50, "Hello, JavaFX!");

Text question = **new** Text(120, 80, "How's it going?");

The start() method creates two objects of the *Text* class – hello and question. The parameters represent x-axis, y-axis, and a string.

Group root = **new** Group(hello, question);

Scene scene = **new** Scene(root, 300, 120, Color.***LIGHTGREEN***);

Those two objects are added to a Group, which is then added to a Scene. The constructor for *Scene* class takes four parameters. The first one is the root node to be displayed, then preferred width and height, and finally background color. In this case the root element is a *Group* object which is to be displayed in an area that is 300 pixels wide and 120 pixels high.

primaryStage.setTitle("A JavaFX Program");

primaryStage.setScene(scene);

primaryStage.show();

On the object primaryStage, we call the setTitle() method to add title displayed on the window’s title bar, setScene(scene) method to set the scene to be displayed on the window and the show() method to display the window.

*launch*(args);

It calls the launch() method to launch the application. Some IDEs may do it automatically.

# Theatre metaphor in JavaFX:

Let’s think about a live Broadway show played on a *Stage*. When the screen goes up a *Scene* is played on the stage and at the end of it the screen comes down. It goes up again for the next scene.

JavaFX uses this metaphor, a Stage is a window. The primary Stage object is created automatically and passed into start() method. A program can make use of multiple stages if desired.

A scene displays a single element here, often referred to as the root displayed. A root node may contain child nodes and grandchild nodes to complete one scene.

# Example 2:

Write a JavaFX application that displays the Olymoic logo.

**import** javafx.application.Application;

**import** javafx.stage.Stage;

**import** javafx.scene.Group;

**import** javafx.scene.Scene;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Circle;

**public** **class** OlympicLogo **extends** Application {

// Presents the Olympic five-ring logo.

**public** **void** start(Stage primaryStage) {

Circle blueRing = **new** Circle(180, 70, 30);

blueRing.setStroke(Color.***BLUE***);

blueRing.setStrokeWidth(6);

blueRing.setFill(**null**);

Circle orangeRing = **new** Circle(215, 100, 30);

orangeRing.setStroke(Color.***ORANGE***);

orangeRing.setStrokeWidth(6);

orangeRing.setFill(**null**);

Circle blackRing = **new** Circle(250, 70, 30);

blackRing.setStroke(Color.***BLACK***);

blackRing.setStrokeWidth(6);

blackRing.setFill(**null**);

Circle greenRing = **new** Circle(285, 100, 30);

greenRing.setStroke(Color.***GREEN***);

greenRing.setStrokeWidth(6);

greenRing.setFill(**null**);

Circle redRing = **new** Circle(320, 70, 30);

redRing.setStroke(Color.***RED***);

redRing.setStrokeWidth(6);

redRing.setFill(**null**);

Group root = **new** Group(blueRing, orangeRing, blackRing, greenRing,

redRing);

Scene scene = **new** Scene(root, 500, 200, Color.***WHITE***);

primaryStage.setTitle("Olympic Logo");

primaryStage.setScene(scene);

primaryStage.show();

}

**public** **static** **void** main(String[] args)

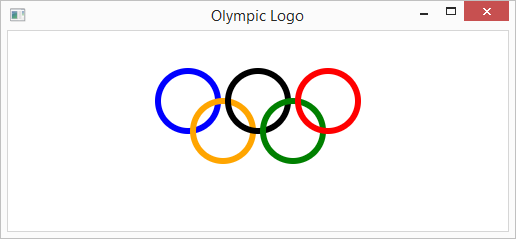
{

*launch*(args);

}

}

Run the program and you will get the following output:



## Description:

The class *Shape* and all its methods are displayed on this page: [Shape (JavaFX 8)](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape.html)

# Example 3:

Write a JavaFX application that displays a rectangle filled with a random color created using the RGB method of the Color class. Each time the program is run it will display a rectangle with a different color.

**import** java.util.Random;

**import** javafx.application.Application;

**import** javafx.stage.Stage;

**import** javafx.scene.Group;

**import** javafx.scene.Scene;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Rectangle;

**public** **class** RandomRectangleColor **extends** Application {

//Presents a rectangle filled with a random color.

**public** **void** start(Stage primaryStage) {

Random gen = **new** Random();

**int** red = gen.nextInt(256);

**int** green = gen.nextInt(256);

**int** blue = gen.nextInt(256);

Rectangle rect = **new** Rectangle(50, 50, 300, 100);

rect.setFill(Color.*rgb*(red, green, blue));

Group root = **new** Group(rect);

Scene scene = **new** Scene(root, 400, 200, Color.***WHITE***);

primaryStage.setTitle("Random Rectangle Color");

primaryStage.setScene(scene);

primaryStage.show();

}

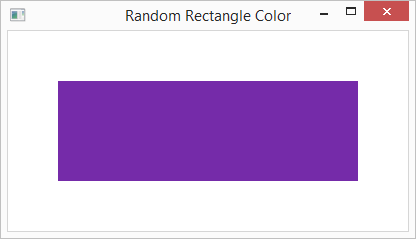
**public** **static** **void** main(String[] args){

*launch*(args);

}

}

Run the program and you will get the following output. Each time you run it, the fill color of the rectangle is different.



## Description:

The only new line of code here is:

rect.setFill(Color.*rgb*(red, green, blue));

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 alpha. The value of alpha is proportional to the opacity of the color. The alpha value 0 means that the color is completely transparent while the value 1 means that the color is completely opaque.

# Example 4:

Write a JavaFX application in which we are inserting a Circle, Sphere and a Text in the same order (that means three nodes) in that order.

We will use the StackPane class which is one of several layout panes provided with JavaFX API. A layout pane is a container that governs how controls are arranged and presented visually. In StackPane, the nodes are arranged one on top of another, just like in stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it.

The class **StackPane** contains a single property named alignment. This property represents the alignment of the nodes within the stack pane. In addition to these, this class also provides a method named **setMargin()**. This method is used to set margin for the node within the stack pane.

**import** javafx.application.Application;

**import** javafx.collections.ObservableList;

**import** javafx.geometry.Insets;

**import** javafx.scene.Scene;

**import** javafx.scene.layout.StackPane;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Circle;

**import** javafx.scene.shape.Sphere;

**import** javafx.scene.text.Font;

**import** javafx.scene.text.FontWeight;

**import** javafx.scene.text.Text;

**import** javafx.stage.Stage;

**public** **class** StackPaneExample **extends** Application {

@Override

**public** **void** start(Stage stage) {

//Drawing a Circle

Circle circle = **new** Circle(300, 135, 100);

circle.setFill(Color.***DARKSLATEBLUE***);

circle.setStroke(Color.***BLACK***);

//Drawing Sphere

Sphere sphere = **new** Sphere(50);

//Creating a text

Text text = **new** Text("The world of JavaFX");

//Setting the font of the text

text.setFont(Font.*font*(**null**, FontWeight.***BOLD***, 15));

//Setting the color of the text

text.setFill(Color.***CRIMSON***);

//setting the position of the text

text.setX(20);

text.setY(50);

//Creating a Stackpane

StackPane stackPane = **new** StackPane();

//Setting the margin for the circle

stackPane.*setMargin*(circle, **new** Insets(50, 50, 50, 50));

//Retrieving the observable list of the Stack Pane

ObservableList list = stackPane.getChildren();

//Adding all the nodes to the pane

list.addAll(circle, sphere, text);

//Creating a scene object

Scene scene = **new** Scene(stackPane);

//Setting title to the Stage

stage.setTitle("Stack Pane Example");

//Adding scene to the stage

stage.setScene(scene);

//Displaying the contents of the stage

stage.show();

}

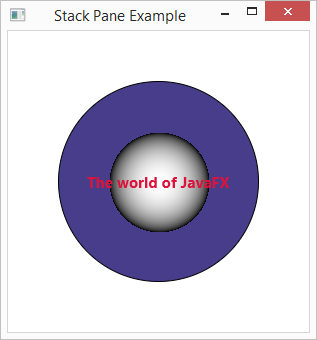
**public** **static** **void** main(String args[]){

*launch*(args);

}

}

Run the program and you will get the following output. Note the superimposed text.



## Description:

We are using StackPane class here:

//Creating a Stackpane

StackPane stackPane = **new** StackPane();

Unlike the previous examples, which used Group object as the root node of the scene, this program displays the image in a StackPane object. It’s a layout pane.

# Example 5:

Write a JavaFX application that presents two buttons and a number (initially 50) to the user. Label the buttons Increment and Decrement. When the Increment button is pushed, increment the displayed value. Likewise, decrement the value when Decrement button is pushed.

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.text.Text;

**import** javafx.scene.layout.FlowPane;

**import** javafx.stage.Stage;

**public** **class** IncrementDecrement **extends** Application {

**private** Text numText;

**private** **int** num;

//Presents two buttons that increment or decrement the value of a

//displayed number.

**public** **void** start(Stage primaryStage) {

num = 50;

numText = **new** Text("50");

Button incButton = **new** Button("Increment");

incButton.setOnAction(**this**::processIncButtonPress);

Button decButton = **new** Button("Decrement");

decButton.setOnAction(**this**::processDecButtonPress);

FlowPane root = **new** FlowPane(incButton, numText, decButton);

root.setAlignment(Pos.***CENTER***);

root.setHgap(20);

root.setStyle("-fx-background-color: beige");

Scene scene = **new** Scene(root, 400, 100);

primaryStage.setTitle("Increment Decrement");

primaryStage.setScene(scene);

primaryStage.show();

}

//Increments the number and updates the text.

**public** **void** processIncButtonPress(ActionEvent event) {

num++;

numText.setText("" + num);

}

//Decrements the number and updates the text.

**public** **void** processDecButtonPress(ActionEvent event) {

num--;

numText.setText("" + num);

}

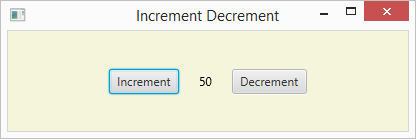
**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

Run the program and you will get the following:



## Description:

Here is a new code:

incButton.setOnAction(**this**::processIncButtonPress);

The :: operator is used to specify a method reference which is available since Java8.

decButton.setOnAction(**this**::processDecButtonPress);

The setOnAction() method sets the value of the property onAction. It’s the EventHandler<Action Event>

root.setHgap(20);

**hgap** − This property is of the type double and it represents the horizontal **gap** between columns.