Intro to Java (Page 8)

JUMP June 2019

Draft: 07/24/2019

Table of Contents

JavaFX

1. Intro
2. Swing vs JavaFX
3. JavaFX Packages
4. The Stage and Scene Classes
5. Nodes and Scene Graphs
6. Layouts
7. The Application Class and the Life-cycle Methods
8. Launching a JavaFX application
9. STS and JavaFX Settings
10. “Hello World” JavaFX Style!
11. The story of 2 buttons and a label
12. 2 Buttons, a Label and some Alignment
13. Let’s Draw!
14. Smarter Event Handling
15. Control Demos

JavaFX - Intro

- JavaFX is designed as a replacement for Swing
- Swing will be part of Java programming for some time to come because of the large amount of legacy code.
- JavaFX is positioned as the platform of the future.
- In the next few years, JavaFX will supplant Swing for new projects.
- Many Swing-based applications will migrate to JavaFX.

- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm

Starting with JDK11, JavaFX is now available as a separate package.

JavaFX – Swing vs JavaFX

While JavaFX has similarities with Java's other GUIs, the AWT and Swing, it has substantial differences.

Similarities

Like Swing, JavaFX components are lightweight and events are handled in an easy-to-manage, straightforward manner.

Differences

The overall organization of JavaFX and the relationship of its main components differ significantly from either Swing or the AWT.

JavaFX – Java FX Packages

The JavaFX framework is contained in packages that begin with the javafx prefix.
There are more than 30 JavaFX packages in its API library.
Here are four examples:

- **`javafx.application.Application;`**
- **`javafx.stage.Stage;`**
- **`javafx.scene.layout.StackPane;`**
- **`javafx.scene.Scene;`**

We will only use a few JavaFX packages in this session, enough to give you a good foundation on how to use JavaFX and further your education should the need arise.

JavaFX – The Stage and Scene Classes

Central to JavaFX's core architecture is the concept and implementation of stage and scene.

All JavaFX apps must have at least one stage and that stage must contain at least one scene.

These elements are encapsulated in the JavaFX API by the Stage and Scene classes.

Stage is a top-level container.

- All JavaFX applications automatically have access to one Stage, called the primary stage.
- The primary stage is supplied by the run-time system when a JavaFX application is started.
- Although you can create other stages, for many applications, the primary stage will be the only one required.

Scene is a container for the items that comprise the scene.

- These can consist of controls, such as push buttons and check boxes, text, and graphics.
- To create a scene, you will add those elements to an instance of Scene.

JavaFX – Nodes and Scene Graphs

- The individual elements of a scene are called **nodes**. For example, a push button control is a node.
- However, **nodes** can also consist of **groups of nodes.**
- Furthermore, a **node** can have a **child node**.
- In this case, **a node** with a **child** is called a **parent node** or **branch node.**
- Nodes without children are **terminal nodes** and are called **leaves**.
- The **collection of all nodes in a scene** creates what is referred to as a **scene graph**, which comprises a **tree**.
- There is one special type of node in the scene graph, called the **root node**.
- This is the top-level node and is the only node in the scene graph that does not have a parent.
- With the exception of the root node, all other nodes have parents, and all nodes either directly or indirectly descend from the root node.
- The base class for all nodes is **Node**. There are several other classes that are, either directly or indirectly, subclasses of Node.
- These include Parent, Group, Region, and Control, to name a few…

JavaFX – Layouts

- JavaFX provides several layout panes that manage the process of placing elements in a scene.
- The **FlowPane** class provides a **flow layout** and the **GridPane** class supports a **row/column grid-based layout**.
- Several other layouts, such as **BorderPane** (which is similar to the AWT's BorderLayout), are available.
- Each inherits Node. The layouts are packaged in **`javafx.scene.layout`**.

JavaFX – The Application Class and the Life-cycle Methods

- A JavaFX application must be a subclass of the **Application** class, which is packaged in **javafx.application**.
- Your application class will extend **Application**.
- The Application class defines **three life-cycle methods** that your application can override.
- These are called **init( )**, **start( )**, and **stop( )**, and they are called in the order presented.

JavaFX – Launching a JavaFX application

- To start a free-standing JavaFX application, you must call the launch( ) method defined by Application.
  - **`public static void launch(String … args)`**
- Here, **args** is a possibly empty list of strings that typically specify command-line arguments.
- When called, **launch( )** causes the application to be constructed, followed by calls to **init( )** and **start( )**.
- The **launch( )** method will not return until after the application has terminated.
- The **launch( )** method starts the subclass of **Application** from which **launch( )** is called.

JavaFX – STS and JavaFX Settings

1) Download Link

Link to SDKs https://gluonhq.com/products/javafx/

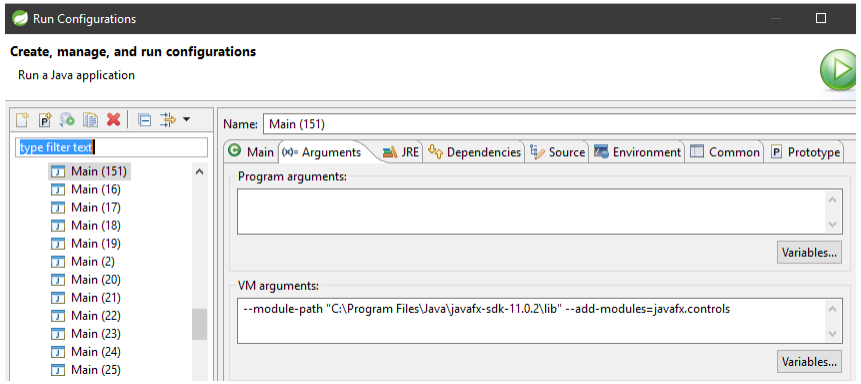
- Download Java FX (SDK) version 11.0.2 (public version)
- Unzip the SDK somewhere you can refer to (notice it will have a javafx-sdk-11.0.2\lib folder
- There are several versions depending on OS.
  (do not download jmods)*

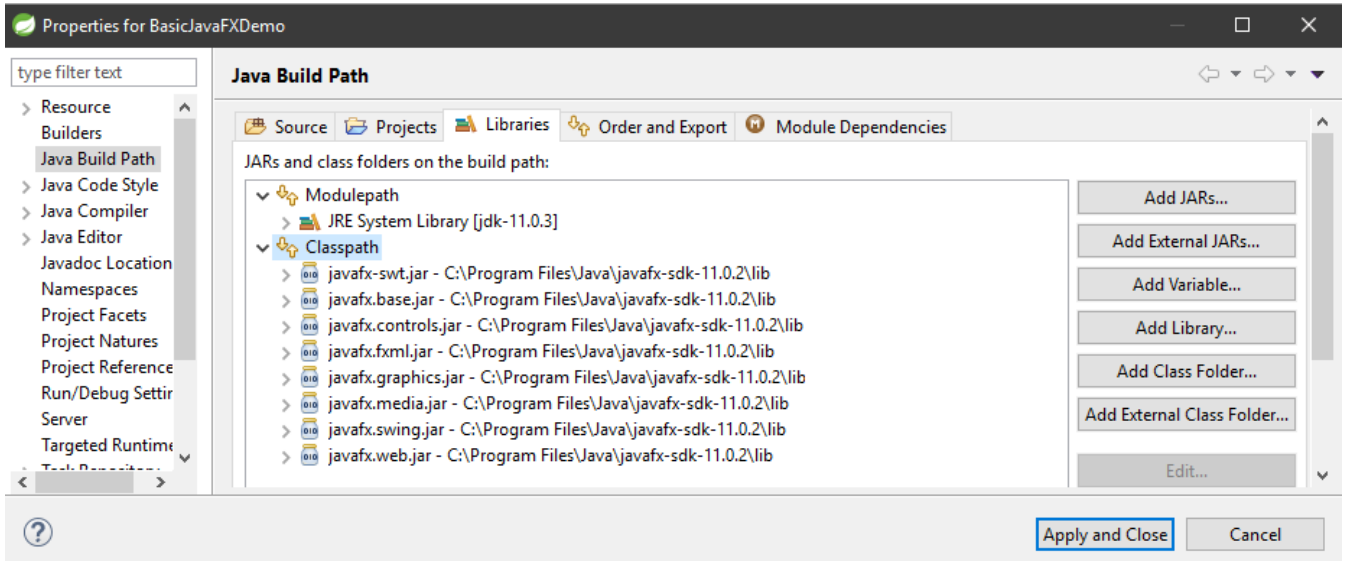
2) When ready to use it in any given project

1. Create a Java Project (STS)
2. Right-click on project (access Properties)
3. Locate Java Build Path
4. Click the Libraries tab, Select the Classpath folder
5. Click Add external JARs
6. Locate the javafx-sdk-11.0.2\lib folder
7. add all JARs from the lib folder
8. Apply and Close

3) Run Configurations - Arguments - VM Options for STS in Windows

```
--module-path "C:\Program Files\Java\javafx-sdk-11.0.2\lib" --add-modules=javafx.controls
```





For those curious about jmods file format: https://stackoverflow.com/questions/44732915/why-did-java-9-introduce-the-jmod-file-format

JavaFX – "Hello World" JavaFX Style!

- Launch() from Application
- Method init()
- Method start()
- Method stop() when window is closed

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.Scene;
```

```
public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    public void init() {
       System.out.println("init() method called first.");
    }

    public void start(Stage primaryStage) {
       System.out.println("start() method called second.");
       primaryStage.setTitle("Hello World!");
         Button btn = new Button();
         btn.setText("Say 'Hello World'");
         btn.setOnAction(new EventHandler<ActionEvent>() {

             public void handle(ActionEvent event) {
                 System.out.println("Hello World!");
             }
         });

         StackPane root = new StackPane();
         root.getChildren().add(btn);
         primaryStage.setScene(new Scene(root, 300, 250));
         primaryStage.show();
    }

    public void stop() {
       System.out.println("stop() method called last when app is closed.");
    }
}
```

In this demo, we have a Hello World panel with an interactive Button
We also get to witness the full lifecycle of the app.
The method stop() will execute when we close the app.

See live code in STS "BasicJavaFXDemo"

JavaFX – The story of 2 buttons and a label

- myLabel
- Button btn and btn2
- Each have their action
- Vbox layout

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.control.*;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;
import javafx.scene.layout.VBox;
import javafx.scene.Scene;;
```

```
public class Main extends Application {

    public void start(Stage primaryStage) {
       primaryStage.setTitle("2 Buttons and a Label");
       Label myLabel = new Label("");
       Button btn = new Button();
       btn.setText("Say 'Hello World'");
       Button btn2 = new Button();
       btn2.setText("Reset");
       btn.setOnAction(new EventHandler<ActionEvent>() {
           public void handle(ActionEvent event) {
            myLabel.setText("hello world!");
           }
       });

       btn2.setOnAction(new EventHandler<ActionEvent>() {
           public void handle(ActionEvent event) {
            myLabel.setText("");
           }
       });
       VBox vbox = new VBox(btn, myLabel, btn2);
       Scene scene = new Scene(vbox, 200, 100);
       primaryStage.setScene(scene);
       primaryStage.show();
    }

}
```

In this demo, we have a label being controlled by 2 buttons

See live code in STS "BasicJavaFXDemo2"

JavaFX – 2 Buttons, a Label and some Alignment

- FlowPane with HGap and VGap
- Geometry.Pos.CENTER
- addAll()

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.control.*;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;
import javafx.scene.layout.FlowPane;
import javafx.scene.Scene;
import javafx.geometry.*;
```

Similar to the previous demo, but with better layout control.

```
public class Main extends Application {
    Label response;

…
    public void start(Stage myStage) {
        myStage.setTitle("JavaFX Buttons, Events and Alignment");
        FlowPane rootNode = new FlowPane(10, 10);
        rootNode.setAlignment(Pos.CENTER);
        Scene myScene = new Scene(rootNode, 300, 100);
        myStage.setScene(myScene);
        response = new Label("Push a Button!");
        Button btnAlpha = new Button("Alpha");
        Button btnBeta = new Button("Beta");
        btnAlpha.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Alpha was pressed.");
            }
        });

        btnBeta.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Beta was pressed.");
            }
        });

        rootNode.getChildren().addAll(btnAlpha, btnBeta, response);
        myStage.show();

    }
}
```

See live code in STS “BasicJavaFXDemo3”

JavaFX – Let's Draw!

- Graphic Context 2D = Canvas
- Various Ways of Drawing

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
import javafx.scene.shape.*;
import javafx.scene.canvas.*;
import javafx.scene.paint.*;
import javafx.scene.text.*;
```

```
public class Main extends Application {
    GraphicsContext gc;
    Color[] colors = { Color.RED, Color.BLUE, Color.GREEN, Color.BLACK};
    int colorIdx = 0;
…
    public int getRandom(int max){
         return (int) (Math.random()*max);
    }
    public void start(Stage myStage) {
        myStage.setTitle("Draw Directly to a Canvas.");
        FlowPane rootNode = new FlowPane();
        rootNode.setAlignment(Pos.CENTER);
        Scene myScene = new Scene(rootNode, 450, 450);
        myStage.setScene(myScene);
        Canvas myCanvas = new Canvas(400, 400);
        gc = myCanvas.getGraphicsContext2D();
        Button btnChangeColor = new Button("Change Color");
        btnChangeColor.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                colorIdx = getRandom(colors.length);
                gc.setStroke(colors[colorIdx]);
                colorIdx = getRandom(colors.length);
                gc.setFill(colors[colorIdx]);
                gc.strokeLine(0, 0, 200, 200);
                gc.fillText("This is drawn on the canvas.", 60, 50);
                gc.fillRect(100, 320, 300, 40);
            }
        });
        gc.strokeLine(0, 0, 200, 200);
        gc.strokeOval(100, 100, 200, 200);
        gc.strokeRect(0, 200, 50, 200);
        gc.fillOval(0, 0, 20, 20);
        gc.fillRect(100, 320, 300, 40);
        gc.setFont(new Font("Arial", 14.0));
        gc.fillText("This is drawn on the canvas.", 60, 50);
        rootNode.getChildren().addAll(myCanvas, btnChangeColor);
        myStage.show();
    }
}
```

Similar to the previous demo, but with better layout control.

- JavaFX's graphics methods are found in the GraphicsContext class, which is part of java.scene.canvas.
- To obtain a GraphicsContext that refers to a canvas, call getGraphicsContext2D( ).

See live code in STS "BasicJavaFXDemo4"

Curious about AWT and Swing?
https://www.oracle.com/technetwork/java/painting-140037.html

JavaFX – Smarter Event Handling

- One Method for Handling Event
- Setting Up IDs
- Casting of Button

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
import javafx.scene.canvas.*;
import javafx.scene.paint.*;
import javafx.scene.text.*;
```

```
public class Main extends Application {
…
    public void myHandle(Button btn) {
        String id = btn.getId();
        switch(id) {
            case "changeColor":
                colorIdx = getRandom(colors.length);
                gc.setStroke(colors[colorIdx]);
                colorIdx = getRandom(colors.length);
                gc.setFill(colors[colorIdx]);
                gc.fillText("This is drawn on the canvas.", 60, 50);
                gc.fillRect(100, 320, 300, 40);

            default:
                System.out.println("Button Id = " + id);
            break;
        }
    }

    public void start(Stage myStage) {
…
        Button btnChangeColor = new Button("Change Color");
        btnChangeColor.setId("changeColor");
        Button btnTest = new Button("Test Button");
        btnTest.setId("Test");
        btnChangeColor.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                myHandle((Button) ae.getSource());
            }
        });
        btnTest.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                myHandle((Button) ae.getSource());
            }
        });
…
    }
}
```

Similar to the previous demo, demonstrating how we can create better event handlers.

- **Basically, we can set up IDs for our buttons**
- **And we can create easy scaffolding of the setOnAction**
- **We also could have easily enough created lambdas for this.**

See live code in STS “BasicJavaFXDemo5”

JavaFX – Control Demos

Demo	Folder
Loading an Image	BasicJavaFXLoadImage
Label and Image Combo	BasicJavaFXImageLabel
Button and Image Combo	BasicJavaFXButtonImage
Toggle Button	BasicJavaFXToggleButton
Radio Button	BasicJavaFXRadioButton
CheckBox	BasicJavaFXCheckBox
ListView	BasicJavaFxListView
ComboBox	BasicJavaFxComboBox
TextField	BasicJavaFxTextField
ScrollPane	BasicJavaFxScrollPane
TreeView	BasicJavaFxTreeView
GridPane	BasicJavaFxGridPane
Special Effects	BasicJavaFXSpecialEffects

JavaFX – Sample App

A JavaFX Sample App

Case Study: Given what we've learned so far….

See live code in STS “JavaFX…”