Published on **ONJava.com** (http://www.onjava.com/)
http://www.onjava.com/pub/a/onjava/2007/07/27/introduction-to-javafx-script.html
See this if you're having trouble printing code examples

Introduction to JavaFX Script

by Anghel Leonard
08/01/2007

What Is JavaFX?

In the spring of 2007 Sun released a new framework called JavaFX. This is a generic name because JavaFX has two major components, Script and Mobile, and, in the future, Sun will develop more components for it.

The core of JavaFX is JavaFX Script, which is a declarative scripting language. It is very different from Java code, but has a high degree of interactivity with Java classes. Many classes of the JavaFX Script are designed for implementing Swing and Java 2D functionalities more easily. With JavaFX Script you can develop GUIs, animations, and cool effects for text and graphics using only a few straightforward lines of code. And, as a plus, you can wrap Java and HTML code into JavaFX Script.

The second component, JavaFX Mobile, is a platform for developing Java applications for portable devices. It will eventually be a great platform for JavaFX Script, but for now is largely irrelevant to the content of this article.

Some Examples of JavaFX Applications

Before we start learning a new language, let's see some examples of JavaFX code. A good resource for examples can be found at the official JavaFX site. To download the examples, please click on JavaFX Script 2D Graphics Tutorial. After the download is complete just double-click the *tutorial.jnlp* file. In a few seconds you should see something like Figure 1 (if you don't see this image, then you have to configure Java Web Start for the *.jnlp* extension).

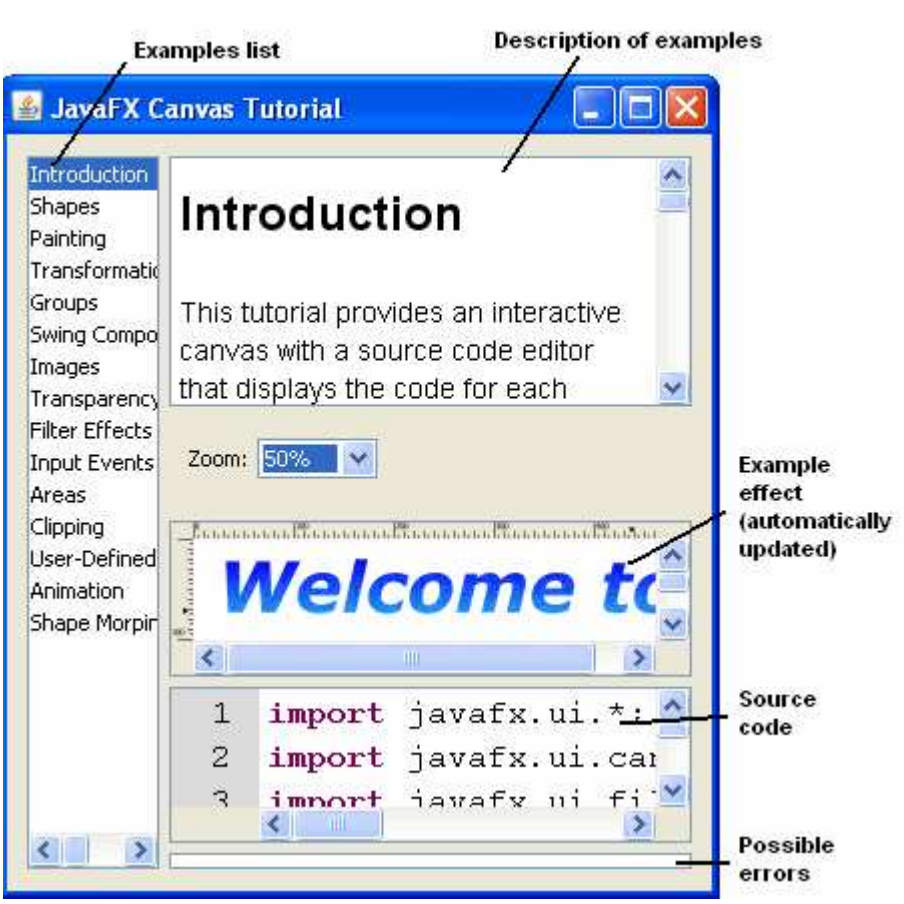

Figure 1. Running the tutorial.jnlp tutorial

Take your time looking over these examples and the source code. There are many interesting effects that can be obtained with just a few JavaFX lines.

If you are still skeptical about the utility of JavaFX, take a look at these two demos; they are partial re-creations of StudioMoto and Tesla Motors sites. You can download them demos from Project OpenJFX by clicking JavaFX Script Studiomoto Demo and JavaFX Script Tesla Demo. They require Java Web Start in order to run, but depending on your machine configuration they may start automatically, or you may have to find and run the downloaded *.jnlp* file.

Download and Install JavaFX

If you are interested in learning to develop JavaFX applications, then you should know that there are at least three methods for working with JavaFX. Also, it is important to know that JavaFX applications are not browser-based. The simplest and quickest method is based on a lightweight tool called JavaFXPad. The major advantage of using this tool is that you can almost immediately see the effect of the changes you are making in the editor. You can download this tool from Project OpenJFX by clicking JavaFX Script JavaFXPad Demo. Again, running this requires Java Web Start (see Figure 2).

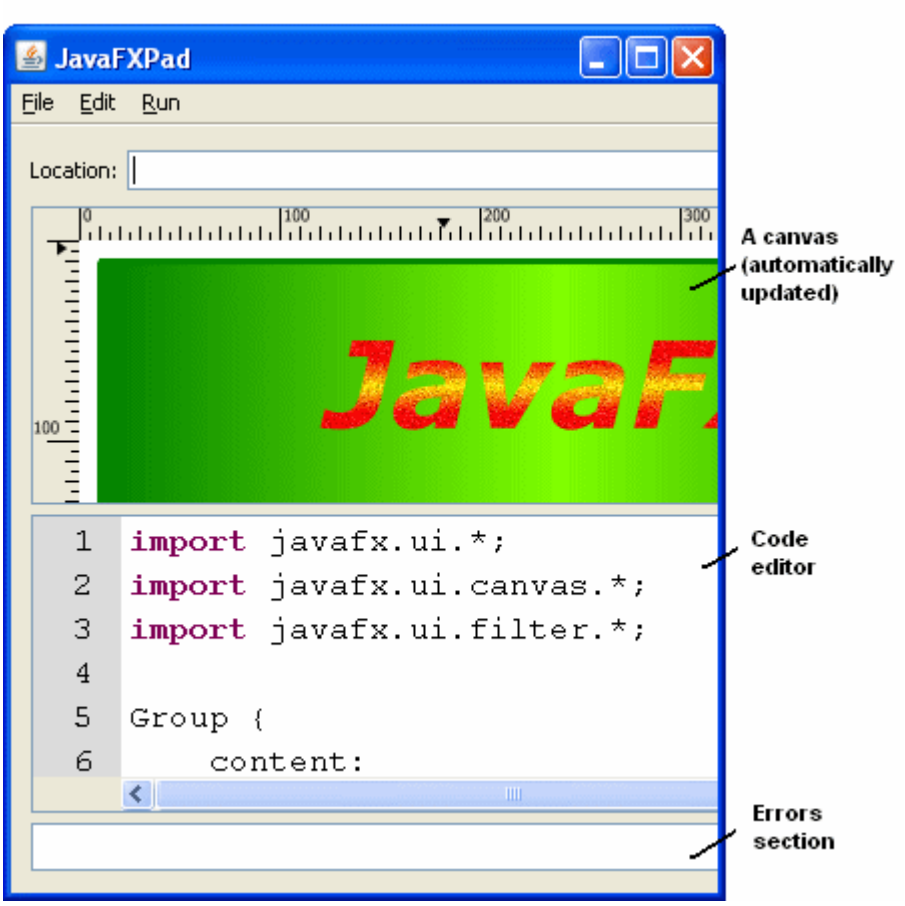

Figure 2. Running the JavaFXPad editor

Another way to work with JavaFX is to use the JavaFX Script Plug-in for NetBeans 5.5 or a JavaFX Script Plug-in for Eclipse 3.2 (of course, before downloading and installing any of these plug-ins you must have NetBeans 5.5 or Eclipse 3.2 already installed).

If you decide to start with the JavaFX plug-in for NetBeans 5.5, the instructions on Project OpenJFX for JavaFX for NetBeans will help you. Similarly, if you want to use the JavaFX plug-in for Eclipse, then go to JavaFX for Eclipse. Notice that all the examples from this article were tested with JavaFX plug-in for NetBeans 5.5, but should work in any of the other listed methods.

Testing the Hello World Application with JavaFX Plug-In for NetBeans 5.5

As always when learning a new language, we have to write the obligatory Hello World application:

Listing 1

```
import javafx.ui.*;
import java.lang.System;
Frame {
    centerOnScreen: true
    visible: true
    height: 50
    width: 350
    title: "HelloWorld application..."
    background: yellow
    onClose: operation() {System.exit(0);}
    content: Label {
        text: "Hello World"
        }
    }
```

To develop and run this simple example in NetBeans 5.5 follow these steps:

1. Launch NetBeans 5.5.
2. From the main menu select File -> New Project.
3. In the New Project window, select the General category and Java Application project (click Next).
4. In the New Java Application window, type "FXExample" in the Project Name text field.
5. In the same window use the Browse button to select the location of the project.
6. Uncheck the "Set as main project" and "Create main class" checkboxes (click Finish).
7. Right-click on the FXExample -> Source Packages and select New -> File/Folder.
8. In the New File window select the Other category and the JavaFX File file type (click Next).
9. In the New JavaFX File window, type "HelloWorld" for File Name and "src" for Folder (click Finish).
10. Copy the code from Listing 1 and paste it in *HelloWorld.fx*.
11. Right-click an *FXExample* project and select Properties.
12. In the Project Properties – FXExample, select the Run node from the Categories pane.
13. In the Arguments text field, type "Hello World" (click OK).
14. Right-click on *FXExample* project and select Run Project option.

If everything works, you should see a frame like in Figure 3:

Figure 3. Running the Hello World application in NetBeans 5.5

Now you have the software support for developing and running any JavaFX application.

JavaFX Syntax

Before starting with JavaFX, let's go over some of the fine points of the syntax. If you are already familiar with the syntax of the Java language, most of this will look very familiar, but some of it is quite different.

JavaFX Primitive Types

JavaFX supports four primitive types: `String` (for `java.lang.String`), `Boolean` (for `java.lang.Boolean`), `Number` (for `java.lang.Number`) and `Integer` (for `byte,short,int,long,BigInteger`).

JavaFX Variables

A JavaFX variable is declared by using the `var` keyword. See the following examples:

```
var x:Number = 0.9;
var name:String = "John";
var y:Integer = 0;
var flag:Boolean = true;

var numbers:Number = [1,2,3,4,5];
```

JavaFX Operators

The well-known Java operators `&&`, `||`, and `!` are displayedlike this in JavaFX:

- Java: `&&`
  JavaFX: `and`
- Java: `||`
  JavaFX: `or`

- Java: `!`
  JavaFX: `not`

JavaFX Functions

JavaFX supports functions. Here are example simple functions with arguments, variable declarations, and a return statement.

```
function taxes(x) {
    var t:Number = (19.0/100.0)*x;
    return t;
    }
```

JavaFX if Statement

In JavaFX you can use conditional blocks by using the `if` statement. Curly braces are required with this statement . If the `else` clause is another `if` statement, then you can skip curly braces:

```
if (place_your_condition_here) {
        //do something
    } else if (place_your_condition_here) {
        //do something
    } else {
        //do something
    }
```

JavaFX while Statement

The `while` statement is similar to `while` in Java. Curly braces are always required with this statement.

```
while (place_your_condition_here)
    {
    //do something
    }
```

JavaFX for Statement

The `for` statement can be used to loop over an interval (intervals are represented using brackets `[]` and the `..` symbol).

```
//i will take the values: 0, 1, 2, 3, 4, 5
for (i in [0..5])
    {
    //do something with i
    }
```

JavaFX Procedures

JavaFX procedures are marked by the `operation` keyword. Here is a simple example:

```
operation startClock() {
do{
    while(true)
        {
        if(seconds>=360)
    {seconds = 0; seconds = [0,6..360] dur 60000 linear;}
        if(minutes>=360)
    {minutes = 0; minutes = [0,6..360] dur 3600000 linear;}
        if(hours>=360)
    {hours = 0; hours = [0,6..360] dur 43200000 linear;}
```

```
        }
    }
}
```

JavaFX Classes

JavaFX classes are marked by the `class` keyword. A JavaFX class may extend more classes by specifying the `extends` keyword and a comma separated list with the names of those classes. Between curly braces you can place attributes, functions, and operations, like in the next example:

```
class Order {
    attribute order_nr: String;
    attribute ordertype: Order inverse Order.products;
    attribute products: Order* inverse Order.ordertype;
    function getOrderNr(): String;
    operation addOrder(order: Order);
    }
```

Notice that attributes are declared using the `attribute` keyword and that the body of the function and procedures are not in the class body. Their compartments are defined after the class declaration, as you will see soon.

The `inverse` clause is optional and it shows a bidirectional relationship to another attribute in the class of the attributes' type. In this case, JavaFX will automatically perform updates (insert, replace, and delete).

You can find a more complete tutorial on Java.net.

Playing Around with JavaFX

In this section, you will see a set of examples that covers a variety of JavaFX possibilities and particularities. The main purpose of these examples is to get you familiar with JavaFX code writing and with the logic of a JavaFX application. The second goal is to convince you that JavaFX deserves a closer look when you need to develop cool GUIs, animations, and nice effects with only a few lines of code. All the examples presented will introduce skills that are specific to JavaFX.

Every example is preceded by a short description, so don't expect line by line comments. All of these should be accessible enough to run yourself, so let's begin.

When you need to print the value of a variable/attribute with `System.out.println` *you can place the name of it inside quoted strings, as shown in Listing 2:*

Listing 2

```
//expressions within quoted text
import java.lang.System;
var mynumber:Number = 10;
System.out.println("Number is: {mynumber}");
```

Result: Number is: 10

JavaFX supports a useful facility known as the cardinality of the variable. This facility is implemented with the next three operators:

- `?`: Optional (may be `null`)
- `+`: One or more
- `*`: Zero or more

Listing 3

```
//cardinality of the variable
import java.lang.System;
var mynumbers:Number* = [1,2,7];
System.out.println("Numbers are: {mynumbers}");
```

Result: Numbers are: 1 2 7

In JavaFX, it is possible to not specify the type of a variable in its declaration. This will not be an error, because JavaFX will automatically find out the type of the variable from its use.

Listing 4

```
//the variable's type is optional
import java.lang.System;
var days = ["Monday,","Friday,","Sunday"];
System.out.println("You have to work: {days}");
```

Result: You have to work: Monday, Friday, Sunday

To get the size of an array, you can use the `sizeof` *operator:*

Listing 5

```
//getting the size of an array
import java.lang.System;
var lotto = [21,30,11,40,5,6];
System.out.println("Array size:{sizeof lotto}");
```

Result: Array size: 6

To get an array from a subarray that satisfies a condition, you can use the `[]` *operator. The condition is placed between* `[]` *and is evaluated as a Boolean. This is similar with the XPath predicates.*

Listing 6

```
//using the [] operator - similar to its use in XPath
import java.lang.System;
var mynumbers = [1,2,7,3,30,15,14,6,4];
var numbers = mynumbers[n|n < 10];
System.out.println("Numbers smaller that 10 are: {numbers}");
```

Result: Numbers smaller than 10 are: 1 2 7 3 6 4

To get the ordinal position of an element within an array, you can use the `indexof` *operator:*

Listing 7

```
//returning the ordinal position of an element within an array
import java.lang.System;
var mynumbers = [1,2,7,3,30,15,14,6,4];
var number_four = mynumbers[indexof . == 4];
System.out.println("Number four:{number_four}");
```

Result: Number four: 30

When you want to insert an element into an array you can use the `insert` *statement and one of the following:*

- `as first`: for inserting into the first position
- `as last`: for inserting into the last position (this is the default)

- `before`: for inserting before a position
- `after`: for inserting after a position

When you want to delete an element from an array, you can use the `delete` *statement.*

Listing 8

```
//insert and delete statement
import java.lang.System;
var mynumbers = [1,2,7];
System.out.println("Before inserting anything:
    {mynumbers}");
insert 10 into mynumbers;
System.out.println("After inserting at the end
    the \"10\" value:{mynumbers}");
insert [8,6,90] as first into mynumbers;
System.out.println("After inserting at the first
    positions the \"8,6,90\" values:{mynumbers}");
insert 122 as last into mynumbers;
System.out.println("After inserting at the
    end the \"122\" value:{mynumbers}");
insert 78 before mynumbers[3];
insert 11 after mynumbers[3];
System.out.println("After inserting the \"78\"
    and \"11\" values before/after the 3rd
    element:{mynumbers}");
delete mynumbers[. == 122];
System.out.println("After deleting:{mynumbers}");
```

Result:
Before inserting anything: 1 2 7
After inserting the 10 value at the end: 1 2 7 10
After inserting the 8, 6, and 90 values at the first positions: 8 6 90 1 2 7 10
After inserting the 122 value at the end: 8 6 90 1 2 7 10 122
After inserting the 78 and 11 values before/after the 3rd element: 8 6 90 78 11 1 2 7 10 122
After deleting: 8 6 90 78 11 1 2 7 10

A great facility of JavaFX is its list comprehensions. This facility is implemented by the `select` *and* `foreach` *operators. Here are two examples (one with* `select` *and one with* `foreach`*) for getting the odd numbers from an interval.*

Listing 9

```
//JavaFX select and foreach operators
import java.lang.System;
function odd(p:Number) {
    return select i from i in [1.0 ..p]
        where (i%2 == 0.0);
    }
var result = odd(10.0);
System.out.println("Odd numbers:{result}");
```

Result: Odd numbers: 2.0 4.0 6.0 8.0 10.0

Listing 10 (same as Listing 9, but with foreach)

```
//JavaFX select and foreach operators
import java.lang.System;
function odd(p:Number) {
    return foreach (i in [1.0 ..p] where (i%2 == 0.0)) i;
    }
```

```
var result = odd(10.0);
System.out.println("Odd numbers:{result}");
```

This example illustrates that `foreach` *can be very useful for creating nice effects.*

Listing 11

```
//JavaFX select and foreach operators
import java.lang.*;
import javafx.ui.*;
import javafx.ui.canvas.*;
Frame {
    centerOnScreen: true
    visible: true
    height: 500
    width: 500
    title: "Foreach demo..."
    onClose: operation() {System.exit(0);}
    content: ScrollPane {
    background: white
    view: Canvas {
        content: bind foreach (i in [1..8], j in [1..8])
        Rect {
        x: i*30
        y: j*30
        width:30
        height:30
        fill: Color {red: (100+i) green: (100+j) blue: (100+(i*j))}
        stroke:white
        strokeWidth:1
        }
    }
    }
}
```

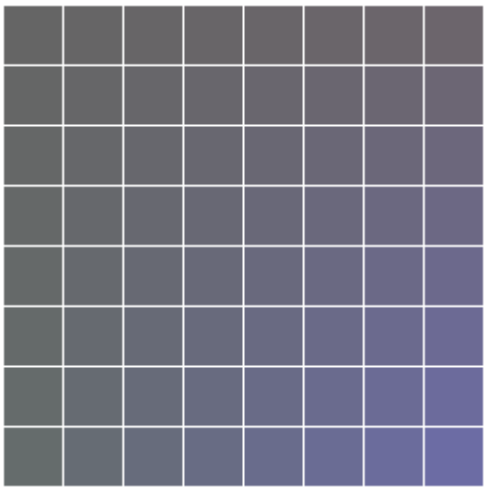

Figure 4. Running Listing 11

When you need to use a variable or attribute that has the same name as a JavaFX keyword you must insert that name between double angle brackets, like this:

Listing 12

```
//Identifier quotes
import java.lang.System;
for (<<for>> in [0..3]) {
    System.out.println("for = {<<for>>}");
    }
```

Result: for = 0 for = 1 for = 2 for = 3

JavaFX can be a great tool when you need to develop Swing interfaces, because JavaFX considerably reduces the amount of code and cooperates well with the `javax.swing.*` *packages. In an earlier section (Testing the Hello World application with JavaFX plug-in for NetBeans 5.5), you saw how easy it is to create a simple frame. Here are two more examples, creating a* `Button` *and a* `TextField`*:*

Listing 13

```
import javafx.ui.*;
import java.lang.System;
Frame{
    content: Button {
    text: "Exit"
    action: operation() {
    System.exit(0);
    }
    }
    visible: true
}
```

Figure 5. Running Listing 13

Listing 14

```
import javafx.ui.*;
Frame {
    content: GroupPanel {
    var myRow = Row { alignment: BASELINE }
    var label_col = Column { alignment: TRAILING }
    var field_col = Column { alignment: LEADING  }
    rows: [myRow]
    columns: [label_col, field_col]
    content:
    [SimpleLabel {
    row: myRow
    column: label_col
    text: "Type your text here:"
    },
    TextField {
    row: myRow
    column: field_col
    columns: 50
    }]
    }
    visible: true
};
```

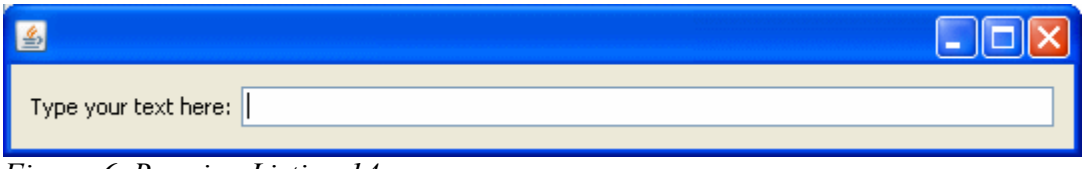

Figure 6. Running Listing 14

Java.net also has a tutorial on creating Swing interfaces with JavaFX.

JavaFX code can be easily integrated with Java code. Here is an example that uses JavaFX to load an image into a frame and allows the user to select a rectangular zone and save it. The capture and save operations are accomplished using Java code.

Listing 15

```
import java.io.*;
import javafx.ui.*;
import javafx.ui.canvas.*;
import javafx.ui.filter.*;
import java.awt.Robot;
import java.awt.Rectangle;
import java.awt.image.RenderedImage;
import javax.imageio.ImageIO;
import java.lang.System;
class CaptureExample extends CompositeNode{
attribute lx: Integer;
attribute ly: Integer;
operation CaptureExample();
}
attribute CaptureExample.lx = 0;
attribute CaptureExample.ly = 0;
operation saveCapture(lx_copy:Integer, ly_copy:Integer) {
var robot = new Robot();
var rect = new Rectangle (lx_copy, ly_copy, 50, 50);
var BI=robot.createScreenCapture(rect);
var file = new File(".//capture.jpg");
ImageIO.write((RenderedImage)BI, "jpg", file);
}
function CaptureExample.composeNode() =
Group{
    transform: []
    content:[ImageView {
    transform: []
    image: Image { url: ".//app//Sunset.gif" }
    cursor: DEFAULT
    onMouseClicked: operation(e:CanvasMouseEvent) {
    saveCapture(e.source.XOnScreen,e.source.YOnScreen);
    }
    onMouseMoved: operation(e:CanvasMouseEvent) {
    lx = e.x;
    ly = e.y;
    }
    },
    Rect{
    x: bind lx
    y: bind ly
    width: 50
    height:50
    strokeWidth: 1
    stroke: black
    }]
    };
Frame {
    centerOnScreen: true
    visible: true
    height: 230
    width: 300
    title: "Capture the screen..."
    onClose: operation() {System.exit(0);}
    content: ScrollPane {
    background: white
    view: Canvas {
    background: black
    cursor: DEFAULT
    content: CaptureExample
    }
   }
}
```

Notice the use of `bind`*. This is a very important JavaFX operator used for incremental and lazy evaluation of attributes. You can find out more about this operator in the JavaFX Programming*

Language documentation.

Also, notice that, it is possible to interact in the above application using two mouse events: mouse clicked (`onMouseClicked`) and mouse moved (`onMouseMoved`). JavaFX supports the following mouse events:

- `onMouseClicked`
- `onMouseMoved`
- `onMousePressed`
- `onMouseExited`
- `onMouseEntered`
- `onMouseReleased`
- `onMouseDragged`

For asynchronous execution of code you can use the JavaFX `do later` statement like this:

Listing 16

```
//asynchronous execution with do later statement
import java.lang.System;
var s1 = "My name is ";
var s2 = "Anghel Leonard";
    do later {
    System.out.println(s2);
    }
System.out.println(s1);
```

Result: My name is Anghel Leonard

JavaFX allows you to execute a portion of code in a separate thread by placing that code into a `do` statement. Using this technique the AWT Event Dispach Thread will be able to process all the incoming events. Here is an example that uses an infinite loop into a `do` statement. Notice that even if you have a infinite loop you still can close the window normally.

Listing 17

```
import javafx.ui.*;
import java.lang.System;
import javafx.ui.canvas.*;
import java.util.Random;
class DoExample extends CompositeNode{
attribute randomfill: Color;
operation changeOpacity();
}
attribute DoExample.randomfill = Color{red:0 green:0 blue:0};
operation DoExample.changeOpacity() {
do{
    while(true)
    {
    var r = new Random();
    var g = r.nextInt(255);
    randomfill = Color{red:g green:g blue:g};
    }
}
}
function DoExample.composeNode() =
Group {
    transform: []
    content: [
    Text {
    x: 20
    y: 20
    content: "Because of \"do\" you can close this window..."
    font: Font {face: VERDANA, style: [ITALIC, BOLD], size: 20}
```

```
    fill: bind randomfill
    opacity: 0.5
    onMouseClicked: operation(e:CanvasMouseEvent) {
    changeOpacity();
    }
    }]
};
Frame {
    centerOnScreen: true
    visible: true
    height: 100
    width: 580
    title: "\"Do\" example..."
    onClose: operation() {System.exit(0);}
    content: ScrollPane {
    background: white
    view: Canvas {
    background: black
    cursor: DEFAULT
    content: DoExample
    }
    }
}
```

JavaFX code can be integrated with HTML code using the JavaFX `Label` *class. This class supports HTML and CSS in the same manner as a web application. Here is an example that renders an HTML table.*

Listing 18

```
import javafx.ui.*;
import java.lang.System;
import javafx.ui.canvas.*;
class Partners {
    attribute name: String;
    attribute company: String;
    attribute phone: String;
    attribute e_mail: String;
    attribute partners: Partners*;
    }
    var myPartners = Partners {
    partners: [Partners{
    name: "Mary J"
    company: "Software ATV Inc."
    phone: "0900090345"
    e_mail: "maryj@yahoo.com"
    },
    Partners{
    name: "Leona W"
    company: "Winkle LTD"
    phone: "090849435"
    e_mail: "leonaw@yahoo.com"
    },
    Partners{
    name: "Joe T"
    company: "Press OJ"
    phone: "340909879"
    e_mail: "joet@yahoo.com"
    }]
    };
    Frame {
    content: Label {
    text: bind "<html>
    <h2 align='center'>- My Partners -</h2>
    <table align='center' border='0' bgcolor='#BBAAEE'>
    <tr bgcolor='#FFEE55'>
    <td><b>Name</b></td>
```

```
    <td><b>Company</b></td>
    <td><b>Phone</b></td>
    <td><b>E-mail</b></td>
    </tr>
    {
    if (sizeof myPartners.partners == 0)
    then "<tr bgcolor='#432211'><td colspan='8'><b>
    I have no partners...</b></td></tr>"
    else foreach (t in myPartners.partners)
    "<tr bgcolor='#FF25AD'>
    <td>{t.name}</td>
    <td>{t.company}</td>
    <td>{t.phone}</td>
    <td>{t.e_mail}</td>
    </tr>"
    }
    </table>
    </html>"
    }
    visible: true
    }
```

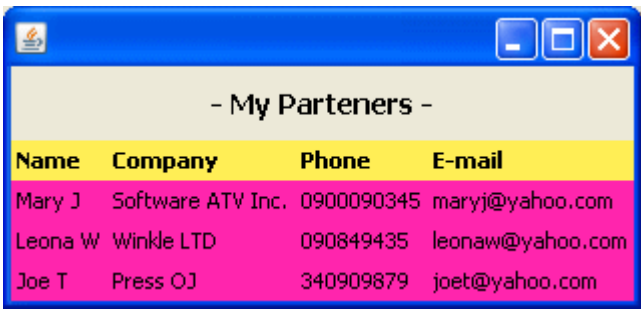

Figure 7. Running Listing 18

In the last section of this article we will see two applications that illustrate how easy it is to build complex animations using JavaFX. The first application simulates an analog clock. The design is based on four images animated with JavaFX code.

Listing 19

```
import javafx.ui.*;
import javafx.ui.canvas.*;
import javafx.ui.filter.*;
import java.lang.System;
class Clock extends CompositeNode{
attribute seconds: Number+;
attribute minutes: Number+;
attribute hours: Number+;
operation startClock();
}
attribute Clock.seconds = 360;
attribute Clock.minutes = 360;
attribute Clock.hours = 360;
operation Clock.startClock() {
do{
    while(true)
    {
    if(seconds>=360) {seconds = [0,6..360]
     dur 60000 linear;}
    if(minutes>=360) {minutes = 0; minutes = [0,6..360]
     dur 3600000 linear;}
    if(hours>=360) {hours = 0;hours = [0,6..360]
     dur 43200000 linear;}
    }
    }
}
function Clock.composeNode() =
```

```
Group {
    onMouseClicked: operation(e:CanvasMouseEvent) {
    startClock();
    }
    transform: []
    content:[ImageView {
    image: Image { url: ".//app//clockempty.gif" }
    },
ImageView {
    transform: bind [translate(203,82),
     rotate (seconds,2.5,125)]
    image: Image { url: ".//app//l1.gif" }
    antialias: true
    },
ImageView {
    transform:  bind [translate(203,100),
     rotate (minutes,2.5,107)]
    image: Image { url: ".//app//l2.gif" }
    antialias: true
    },
ImageView {
    transform:  bind [translate(203,115),
     rotate (hours,2.5,92)]
    image: Image { url: ".//app//l3.gif" }
    antialias: true
    },
Circle {
    cx: 205
    cy: 205
    radius: 13
    fill: red
    strokeWidth: 1
    }]
};
Frame {
    centerOnScreen: true
    visible: true
    height: 450
    width: 450
    title: "JavaFX - Clock"
    onClose: operation() {System.exit(0);}
    content: ScrollPane {
    background: white
    view: Canvas {
    background: black
    cursor: DEFAULT
    content: Clock
    }
    }
}
```

Figure 8. Running Listing 19

Notice the `dur` *(duration) operator. JavaFX provides this operator for creating animations. It is used to apply an array asynchronously to a time interval. Before returning an element from the array JavaFX will wait a time equal to the specified milliseconds. This process will be repeated until the whole elements are returned. There are four types of interpolations:*

- `linear`
- `easein`
- `easeout`
- `easeboth`

By default it is a `ease-in`, `ease-out` *combination.*

The second application simulates a set of pop-up menus that can be dragged on the screen. With a simple click on the menu headers you can hide/show the items using a nice opacity effect. The whole design was written with JavaFX drawing capabilities:

Listing 20

```
import javafx.ui.*;
import javafx.ui.canvas.*;
import javafx.ui.filter.*;
import java.lang.System;
class MenuOptions extends CompositeNode{
attribute px: Integer;
attribute py: Integer;
attribute lx: Integer;
attribute ly:Integer;
attribute lw:Integer;
attribute itemsOpacity:Number;
attribute menutext: String;
}
trigger on new MenuOptions {
    this.px = 0;
    this.py = 0;
    this.menutext = "";
    this.lx = 0;
    this.ly = 0;
    this.lw = 150;
    this.itemsOpacity = 0.0;
    }
function MenuOptions.composeNode() =
Group {
    transform: bind []
    opacity: bind itemsOpacity
    content:[Rect {
    x: bind lx
    y: bind ly
    width: lw
    height: 20
    arcHeight: 10
    arcWidth: 10
    fill: Color {red:190 green:181 blue:215}
    stroke: Color {red:68 green:54 blue:103}
    strokeWidth: 2
    onMouseEntered: operation(e:CanvasMouseEvent) {
    if(itemsOpacity == 0.7) {itemsOpacity = 1.0;}
    }
    onMouseExited: operation(e:CanvasMouseEvent) {
    if(itemsOpacity == 1.0) {itemsOpacity = 0.7;}
    }
    onMouseClicked: operation(e:CanvasMouseEvent) {
    eventListener(this.menutext);
    }
    },
```

```
    Text {
    x: bind lx+5
    y: bind ly+5
    content: bind menutext
    font: Font {face: VERDANA, style: [BOLD], size: 11}
    fill:Color {red:68 green:54 blue:103}
    }]
    };
class MainMenu extends CompositeNode{
attribute option: String;
attribute px: Integer;
attribute py: Integer;
attribute lx: Integer;
attribute ly:Integer;
attribute lw:Integer;
attribute menutext: String;
attribute step: Integer;
attribute submenu: MenuOptions+;
operation addSubmenu(t:MenuOptions);
operation show_hide();
}
trigger on new MainMenu {
    this.option = "";
    this.px = 0;
    this.py = 0;
    this.menutext = "";
    this.lx = 0;
    this.ly = 0;
    this.lw = 150;
    this.step = 20;
    this.submenu = null;
    }
operation MainMenu.addSubmenu(t:MenuOptions) {
t.lx = this.lx;
t.lw = this.lw;
t.ly = this.ly+step;
step=step+20;
insert t into submenu;
}
operation MainMenu.show_hide() {
if(submenu.itemsOpacity[1] == 0.7){submenu.itemsOpacity =
 [0.7,0.6,0.5,0.4,0.3,0.2,0.1,0.0] dur 1200;}
    else if(submenu.itemsOpacity[1] == 0.0){
    submenu.itemsOpacity =
     [0.1,0.2,0.3,0.4,0.5,0.6,0.7] dur 1200;}
}
function MainMenu.composeNode() =
Group {
    transform: bind []
    content:[Rect {
    x: bind lx
    y: bind ly
    height: 20
    width: bind lw
    arcHeight: 10
    arcWidth: 10
    fill: Color {red:68 green:54 blue:103}
    stroke: Color {red:190 green:181 blue:215}
    strokeWidth: 2
    onMouseDragged: operation(e:CanvasMouseEvent) {
    lx += e.localDragTranslation.x;
    ly += e.localDragTranslation.y;
    submenu.lx += e.localDragTranslation.x;
    submenu.ly += e.localDragTranslation.y;
    }
    onMouseClicked: operation(e:CanvasMouseEvent) {
    show_hide();
    }
    },
```

```
    Text {
    x: bind lx+5
    y: bind ly+5
    content: bind menutext
    font: Font {face: VERDANA, style: [BOLD], size: 11}
    fill:Color {red:190 green:181 blue:215}
    },
    bind submenu]
};
var menu_1 = new MainMenu();
menu_1.lx = 120;
menu_1.ly = 140;
menu_1.lw = 128;
menu_1.menutext = "Navigate";
var submenu_11 = new MenuOptions();
submenu_11.menutext = "Go to Class...";
var submenu_12 = new MenuOptions();
submenu_12.menutext = "Go to Test";
var submenu_13 = new MenuOptions();
submenu_13.menutext = "Back";
var submenu_14 = new MenuOptions();
submenu_14.menutext = "Forward";
var submenu_15 = new MenuOptions();
submenu_15.menutext = "Go to Line...";
menu_1.addSubmenu(submenu_11);
menu_1.addSubmenu(submenu_12);
menu_1.addSubmenu(submenu_13);
menu_1.addSubmenu(submenu_14);
menu_1.addSubmenu(submenu_15);
var menu_2 = new MainMenu();
menu_2.lx = 260;
menu_2.ly = 140;
menu_2.lw = 90;
menu_2.menutext = "Refactor";
var submenu_21 = new MenuOptions();
submenu_21.menutext = "Rename....";
var submenu_22 = new MenuOptions();
submenu_22.menutext = "Pull Up...";
var submenu_23 = new MenuOptions();
submenu_23.menutext = "Push Down...";
menu_2.addSubmenu(submenu_21);
menu_2.addSubmenu(submenu_22);
menu_2.addSubmenu(submenu_23);
operation eventListener(s:String) {
System.out.println("You choose:{s}");
}
Frame {
    centerOnScreen: true
    visible: true
    height: 500
    width: 500
    title: "JavaFX - Menu"
    onClose: operation() {System.exit(0);}
    content: ScrollPane {
    background: white
    view: Canvas {
    background: black
    cursor: DEFAULT
    content: [menu_1, menu_2]
    }
    }
}
```

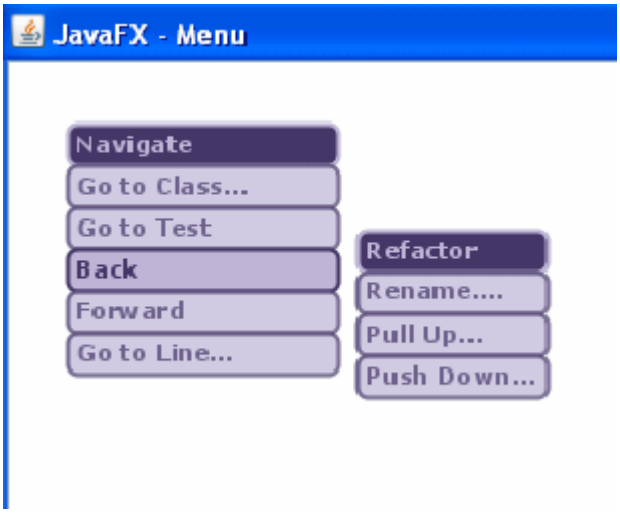

Figure 9. Running Listing 20

Notice that JavaFX uses triggers (like SQL) instead of constructors. A trigger is marked by the `trigger` *keyword and is made of a header and a body. The header indicates the event that must occur before executing the body content. A trigger can be created, inserted, deleted, or replaced. You can find more details about triggers on the OpenJFX site.*

Conclusion

JavaFX Script is a capable new language for the Java platform. With it, you can easily build rich, dynamic interfaces in much less time than you could build something comparable in Java with Swing and Java 2D. In this article, we have stepped through the basic syntax, looked at IDE support, and built a demonstration application that shows off some of JavaFX's Capabilities. JavaFX will quickly become a necessary tool in the Java developer's toolbox.

Resources

- Sample code for this article (for running the examples create a new NetBeans project)
- The offical JavaFX site: from here you can download demos, specifications and plug-ins
- JavaFX classes documentation: get help with JavaFX classes
- Plug-in for NetBeans: get the JavaFX plug-in for NetBeans
- Plug-in for Eclipse: get the JavaFX plug-in for Eclipse

Anghel Leonard is a senior Java developer with more than 12 years of experience, specializing in GIS applications. He has written two books about XML and Java.

Return to ONJava.com.