JavaFX: New Kid on RIA Block

Sang Shin
Sun Microsystems
javapassion.com
Topics

- JavaFX script overview
- Declarative GUI building
- Scene graph
- Animation
- Media
JavaFX Script Overview
JavaFX Script

• Declarative, statically-typed scripting language
• Facilitates rapid GUI development
• Many cool, interesting language features
• Runs on Virtual Machine for the Java™ platform
• Deployment options same as Java programs
• Fully utilizes Java class libraries behind the scenes
• For content designers and Media engineers
Class Definition

• Address class definition

```java
class Address {
    var street: String;
    var city: String;
    var state: String;
    var zip: String;
}
```
Declaring an Object Literal

• In the JavaFX Script programming language, an object instance can be created with an object literal (unlike Java)

• Example:

```java
Address {
    street: "1 Main Street"; // separated by semi colons
    city: "Santa Clara";
    state: "CA";
    zip: "95050";
}
```
Nesting an Object inside Another Object

- Nesting `Address` object inside `Customer` object

```python
def customer = Customer {
    firstName: "John";
    lastName: "Doe";
    phoneNum: "(408) 555-1212";
    address: Address {
        street: "1 Main Street";
        city: "Santa Clara";
        state: "CA";
        zip: "95050";
    }
}
```
Assigning an Object Literal to a Variable

// The variable color is Color type
var color: Color;

var rect = Rectangle {
  x: 10 y: 10 width: 100 height: 100
  fill: color = Color {
    red: 1
    green: 0
    blue: 0
  }
}

var rect2 = Rectangle {
  x: 200 y: 10 width: 100 height: 100
  fill: color
}
What is a Sequence?

- In addition to the five basic data types, the JavaFX Script programming language also provides data structures known as Sequence's.
- A Sequence represents an ordered list of objects; the objects of a sequence are called items.
Creating Sequences

- One way to create a sequence is to explicitly list its items.
- Each element is separated by a comma and the list is enclosed in square brackets [ and ]

> For example, the following code declares a sequence and assigns it to a variable named `weekDays`

> ```javascript
> var weekDays = ["Mon", "Tue", "Wed", "Thu", "Fri"];
> ```

> The compiler knows that we intend to create a "sequence of strings" because the individual items are all declared as String literals
What is a Function?

- A function is an executable block of code that performs a specific task.
- Function code does not execute until it is explicitly invoked.
  > This makes it possible to run a function from any location within your script. It does not matter if the function invocation is placed before or after the function definition.

```javascript
add(); // Invoke add function
function add() {
    result = numOne + numTwo;
    println("{numOne} + {numTwo} = {result} ");
}
```
Passing Arguments to Functions

- Script functions may also be defined to accept arguments.

```javascript
var result;

add(100,10);

// Receive two arguments
function add(argOne: Integer, argTwo: Integer) {
    result = argOne + argTwo;
    println("{argOne} + {argTwo} = {result}");
}
```
Passing Arguments to Functions

- Arguments can be custom types

```swift
// Load image and data specified in given Photo object
function loadImage(photo: Photo, thumbImageView: ThumbImageView): Void {
    thumbImageView.image = Image {
        url: "http://farm{photo.farm}.static.flickr.com/{photo.server}/
        {photo.id}{photo.secret}{imgSuffix}.jpg"
        width: thumbSize
        height: thumbSize
        backgroundLoading: true
        placeholder: thumbImageView.image
    }
    thumbImageView.photo = photo;
}
```
Returning Values from Functions

• A function may also return a value to the code that invokes it

```java
function add(argOne: Integer, argTwo: Integer) : Integer {
    result = argOne + argTwo;
    println("{argOne} + {argTwo} = {result}"):
    return result;
}
```

• The return statement is optional – the below the same

```java
function add(argOne: Integer, argTwo: Integer) : Integer {
    result = argOne + argTwo;
    println("{argOne} + {argTwo} = {result}"):
    return result;
}
```
Binding

• Cause and effect – responding to changes
• `bind` operator allows dynamic content to be expressed declaratively
• Dependency based evaluation of any expression
• Automated by the JavaFX runtime rather than manually wired by the programmer
• Eliminates the listener pattern
Binding

- The `bind` keyword associates the value of a target variable with the value of a bound expression.
- The bound expression can be a simple value of
  - a variable
  - an expression.
  - an object
  - a function
  - a sequence
Binding to a variable

```java
var x = 0;

// Bind variable x to variable y.
def y = bind x;
x = 1;
println(y); // y now equals 1
x = 47;
println(y); // y now equals 47
```

// Because the variables are bound, the value of y
// automatically updates to the new value.
println(y); // y now equals 47
Binding to an Expression

```javascript
var x = 0;

// The variable y is bound to a simple expression x + 10
var y = bind x + 10;

x = 1;
println("----y after x is changed to 1 = {y}"); // y now equals 11

x = 47;
println("----y after x is changed to 47 = {y}"); // y now equals 57
```
Binding to an Object

```java
var myStreet = "1 Main Street";
var myCity = "Santa Clara";
var myState = "CA";
var myZip = "95050";

def address = bind Address {
    street: myStreet;
    city: myCity;
    state: myState;
    zip: myZip;
};

println("address.street == {address.street}"); // prints “1 Main Street”

// By changing the value of myStreet, the street variable inside the address object
// is affected. Note that changing the value of myStreet actually causes a new
// Address object to be created and then re-assigned to the address variable.
myStreet = "100 Maple Street";
println("address.street == {address.street}"); // prints “100 Maple Street”
```
Binding to an Object

```java
var myStreet = "1 Main Street";
var myCity = "Santa Clara";
var myState = "CA";
var myZip = "95050";

// To track changes without creating a new Address object, bind directly to the
// object's instance variables instead:

def address = bind Address {
    street: bind myStreet;
    city: bind myCity;
    state: bind myState;
    zip: bind myZip;
};

println("address.street == {address.street}"); // prints “1 Main Street”
myStreet = "100 Maple Street";
println("address.street == {address.street}"); // prints “100 Maple Street”
```
Definition of a Function

var scale = 1.0;

function makePoint(xPos : Number, yPos : Number) : Point {
    Point {
        x: xPos * scale
        y: yPos * scale
    }
}

class Point {
    var x : Number;
    var y : Number;
}
Invocation of a Bound Function

// Code in the previous slide

// The bind keyword, placed just before the invocation of
// makePoint, binds the newly created Point object (pt) to the outcome of the
// makePoint function.
var myX = 3.0;
var myY = 3.0;
def pt = bind makePoint(myX, myY);
println(pt.x);  // 3.0

myX = 10.0;
println(pt.x);  // 10.0
Binding with Sequences

- We can bind the two sequences by placing the bind keyword just before the for keyword.

```javascript
var seq1 = [1..10];
def seq2 = bind for (item in seq1) item*2;
printSeqs();

function printSeqs() {
    println("First Sequence:");
    for (i in seq1){println(i);}    println("Second Sequence:");
    for (i in seq2){println(i);}  }
```
Result

First Sequence:
1
2
...
9
10

Second Sequence:
2
4
...
18
20
Using Declarative Syntax (for Creating GUI)
Example of JavaFX Application

```java
import javafx.application.*;
import javafx.scene.geometry.*;
import javafx.scene.paint.*;

Stage {
  scene: {
    content: [
      Circle {
        centerX: 50
        centerY: 50
        radius: 50
        fill: Color.RED
      }
    ]
  }
}
```
Why Declarative Syntax for Building GUI?

• Because the structure of declared objects in the code reflects the visual structure of the scene graph, and this enables you to understand and maintain the code easily.

• The order of elements you declare in the code matches the order in which they appear in the application.
Demo:

Building “HelloWorld” JavaFX Application
Lab Work (20 min): 

http://www.javapassion.com/handsonlabs/javafx_lang/#Exercise_0
http://www.javapassion.com/handsonlabs/javafx_lang/#Exercise_1
http://www.javapassion.com/handsonlabs/javafx_lang/#Exercise_2
Scene Graph
What is Scene Graph?

- Scene Graph enables **declarative GUI programming**
- The scene graph is a tree-like data structure which defines a hierarchy of graphical objects in a scene.
- A single element in the scene graph is called a node.
  > Each node has one parent except for the root node, which has no parent.
  > Each node is either a leaf node or a branch.
  > A leaf node has no children.
  > A branch node has zero or more children.
More on Scene Graph Nodes

- JavaFX nodes represent different types of content such as UI components, shapes, text, images, and media.
- Nodes can be transformed and animated.
- You can also apply various effects to nodes.
JavaFX Architecture

JavaFX Script Software

Project Scene Graph

Effects

Java 2D

Graphics hardware

Models a JavaFX GUI
Scene Graph: Group represents Branch

Group {
    transforms: Translate {
        x: 15, y: 15
    }
    content: [
        Text {
            x: 10, y: 50
            font: Font: {
                size: 50
            }
            content: “Hello World”
        }
        Circle {
            centerX: 100, centerY: 100
            radius: 40
            fill: Color.BLACK
        }
    ]
}
Effects
How Effect Works

• Any Effect instance can be applied to a scene graph Node by setting the `Node.effect` variable.

• Each Effect subclass exposes a small number of variables that control the visual appearance of the Node.

• In addition, most Effect subclasses provide one or more input variables (attributes) that can be used to "chain" effects.
Effects:
DropShadow
Example: DropShadow class

- *DropShadow* class provides 5 attributes
  - color: The shadow Color
    - default: Color.BLACK
  - offsetX: The shadow offset in the x direction, in pixels.
    - default: 0.0
  - offsetY: The shadow offset in the y direction, in pixels.
    - default: 0.0
  - radius: The radius of the shadow blur kernel.
    - default: 10.0, max: 63.0
  - spread: The spread of the shadow.
    - default: 0.0, max: 1.0, min: 0.0
Example: DropShadow

Text {
  effect: DropShadow {
    offsetY: 3
    color: Color.color(0.4, 0.4, 0.4)
  }
  ...
}

Circle {
  effect: DropShadow {
    offsetY: 4
  }
  ...
}

JavaFX drop shadow...
Example: DropShadow

Text {
    effect: DropShadow {
        offsetY: 3
        color: Color.GREEN
        radius: 20.0
    };
    ...
},
Circle {
    effect: DropShadow {
        offsetX: 10
        offsetY: 20
        color: Color.BLUE
        radius: 30.0
    }
    ...
}
Example: DropShadow with Binding

- Apply a DropShadow effect to a rounded Rectangle and control its appearance through the magic of the bind operator.

```
Rectangle {
    effect: DropShadow {
        radius: bind radius
    }
    x: 50 y: 30 width: 150 height: 100
    arcWidth: 40 arcHeight: 40
    fill: Color.RED
}
```
Demo:

DropShadow,
DropShadow with Binding,
EffectsPlayground
Lab Work (20 min):

www.javapassion.com/handsonlabs/javafx_guibasics/#Exercise_4
www.javapassion.com/handsonlabs/javafx_guibasics/#Exercise_7
Effects:
PerspectiveTransform
PerspectiveTransform Class

• Used to provide a "faux" three-dimensional effect for otherwise two-dimensional content.

```plaintext
Group {
    effect: PerspectiveTransform {
        ulx: 10 uly: 10 urx: 310 ury: 40
        lrx: 310 lry: 60 llx: 10 lly: 90
    }
    cache: true
    content: [
        Rectangle {
            x: 10 y: 10 width: 280 height: 80 fill: Color.BLUE
        },
        Text {
            x: 20 y: 65 content: "Perspective" fill: Color.YELLOW
            font: Font.font(null, FontWeight.BOLD, 36);
        }
    ]
}
```
Animation Support in JavaFX
Animation Support in JavaFX

- Built in the language syntax
  - Can animate any variable
- Native support for time
  - `Duration` class
  - Time literals – `1ms, 1s, 1m, 1h`
  - Eg. `var runFor = 500ms`
Two Types of Animation in JavaFX

• Transition
  > “Precanned” animation
  > Single purpose

• Animation
  > More flexible but more code
Transitions
Transitions

• Predefined animations to perform a specific task
  > Position, rotation, opacity, etc.

• Out of the box transitions
  > *RotateTransition* – rotation
  > *FadeTransition* – opacity
  > *TranslateTransition* – move a node along a straight line
  > *PathTransition* – move an object along a defined path
  > *ScaleTransition* – grows or shrinks a node
Using Transitions

- Need to specify which node the transition is performed on
  - Nodes – geometric shapes, images, text, Swing components
- Other attributes
  - Duration – how long to perform the animation
  - Rate – the speed and direction
  - Interpolator – the acceleration and deceleration of the animation
- Can execute a function at the end of the animation
  - Assign a function to action attribute
RotationTransition

```javascript
var rotTransition = RotateTransition {
    duration: 3s
    node: node
    byAngle: 180
    repeatCount: 4
    autoReverse: true
}

var princess: ImageView = ImageView {
    image: Image {
        url: "{__DIR__}princess.png"
    }
    onMouseClicked: function( e: MouseEvent ): Void {
        rotTransition.play();
    }
}
```
Path Transition

```javascript
var earth: ImageView = ImageView {
    x: sx y: sy
    image: Image { url: "{__DIR__}earth.png" }
}

def path = [
    MoveTo { x: sx y: sy}
    ArcTo { x: 0 y: 200
        radiusX: 50 radiusY: 50 sweepFlag: true
    }
];

var aniPath: PathTransition = PathTransition {
    node: earth
    path: AnimationPath.createFromPath(
        Path {elements: path })
    duration: 1500ms
}

aniPath.playFromStart();
```
Demo:

Transitions
KeyFrame based Animation
Key Frame based Animation

• What is Key Frame based animation?
  > A declarative model in which programmer describes the animated state transitions of each "scene" by declaring "snapshots" (key frames) of state at certain points in time.

• Two basic varieties of key frame animation
  > Discrete - Set of discrete key frames
  > Interpolated - Special interpolation functions calculate the states that occur between animation frames

• Animation controls
  > Start, stop, pause, and resume
Programming Model of Key Frame Animation

- Animations occur along a timeline, represented by a `javafx.animation.Timeline` object.
- Each timeline contains two or more key frames, represented by `javafx.animation.KeyFrame` objects.
- Each timeline supports
  - Animation attributes
    - `autoReverse`, `repeatCount`, `toggle`, etc.
  - Playback controls
    - `start()`, `stop()`, `pause()`, and `resume()`
Example: Interpolatorator Based

```javascript
var t = Timeline {
    keyFrames : [
        KeyFrame {
            time: 0s
            values: [ tx => 0.0 ]
            action: function() { ... }
        },
        KeyFrame {
            time: 10s
            values: [ tx => 700 tween Interpolatorator.EASEBOTH ]
        }
    ]
}]
}
t.start();
```
Example – Defining Key Frames

Timeline {
    keyFrames: [
        KeyFrame {
            time: 0s
            values: [ radius => 30 ]
        }
        KeyFrame {
            time: 5s
            values: [
                radius => 300 tween Interpolator.LINEAR
            ]
        }
    ]
    Key value
    radius = 30
    How the value changes over time
    radius = 300
}

Keyframes
0s 1s 2s 3s 4s 5s 6s
at() (Shorthand notation)

```javascript
var t = Timeline {
    ...
    keyFrames: [
        KeyFrame {
            time: 0ms
            values: [ radius => 30 ]
        },
        KeyFrame {
            time: 500ms
            values: [
                radius => 300 tween Interpolator.LINEAR
            ]
        }
    ]
    keyFrames: [
        at(0ms) { radius => 30 }
        at(500ms) {
            radius => 300 Interpolate.LINEAR
        }
    ]
}
```
Animation through Binding

```javascript
var opa = 0.0;
var street1:ImageView = ImageView {
    image: Image { url: "{__DIR__}street1.jpg" }
    opacity: bind opa
    onMouseClicked: function( e: MouseEvent ):Void {
        timeline.play();
    }
}

var timeline:Timeline = Timeline {
    keyFrames: [
        KeyFrame {
            time: 0s
            values: [ opa => 0.0,]
        },
        KeyFrame {
            time: 1s
            values: [ opa => 1.0 tween Interpolator.LINEAR,]
        }
    ]
}
```
Lab Work (20 min):

www.javapassion.com/handsonlabs/javafx_animation/#Exercise_1
www.javapassion.com/handsonlabs/javafx_animation/#Exercise_3
Custom Node
Custom Node

• Primary means of Scene Graph encapsulation
  > All other nodes are not extendable
• Use it when extending existing GUI class is not enough for your task
• Simply override the create() method, which returns a Node object
Simple CustomNode

- Extend CustomNode class and override create()

class Bars extends CustomNode {
    override function create():Node {
        return Group {
            content: for(x in [0..4]) {
                Rectangle {
                    y: indexof x * 20
                    width: 100
                    height: 10
                    fill: Color.RED
                }
            }
        }
    }
}

// Bars object literal
Bars { }
Demo:
Building “Picture Display” Step by Step
Demo:

Building “Under the Sea” Step by Step
Lab Work (30 min):

www.javapassion.com/handsonlabs/javafx_animation/index.html#Exercise_6
Motivation and Goals

- Video and audio are ubiquitous on the Net
- Java support is spotty at best – JMF
  - Need to work “out of the box”
- Top grade media support
  - Simple to deploy and program
  - Zero configuration, support whatever the native platform supports
  - Integration with JavaFX platform – scenegraph
Formats, Codecs and Platform Support

- Cross platform video format support
  - Encode once, play anywhere
  - License codec from On2 Technologies

- Leveraging the native platform
  - Windows
    - Play windows media via DirectPlay
    - Flash via the ActiveX control
  - Mac
    - CoreAudio and CoreVideo
  - Solaris and Linux
    - Any audio/video supported by totem can be played
    - May have to recompile gstreamer on some Linux platform
Media Classes

- Media – represents the media source
  - Tracks – audio, video and subtitles currently supported
  - Duration, size, etc
  - Metadata information
- MediaPlayer – controls for playing media
- MediaView – display for MediaPlayer
- Access to events and exceptions
Example of Creating a Media Player

```javascript
var video:Media = Media {
    source: "http://..."
};

var player:MediaPlayer = MediaPlayer {
    media: video
    rate: 1.0
    volume: 0.4
};

var view:MediaView = MediaView {
    mediaPlayer: player
    x:200
    y:200
};

Stage {
    title: "Media Player"
    width: 700
    height: 700
    scene: Scene {
        content: [view]
    }
}
```
Demo: Media
Lab Work (20 min):

http://www.javapassion.com/handsonlabs/javafx_media/#Exercise_1
JavaFX Technology Overview