JavaFX: New Kid on RIA Block

Sang Shin
Sun Microsystems
javapassion.com
Topics

• JavaFX script overview
• Declarative GUI building
• Scene graph
• Animation
• Media
• Deployment
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: The Address class declares street, city, state, and zip instance variables all of type String

```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: The first word (Address) specifies the type of object, class, that you are creating.

```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";
    }
}
```
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 to a Simple Expression

```javascript
var x = 0;

// Bind variable x to variable y. Whenever the value of x changes, the value of variable y automatically changes as well.
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
```
Definition of a Bound Function

var scale = 1.0;

// makePoint is a bound function. It will be invoked even when a value of
// non-function-arugment such as scale changes. If you remove bound
// keyword, then, the change of the value of scale does not invoke the
// function.
bound 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

scale = 2.0;
println(pt.x);  // 20.0
What is Trigger?

- Allows a block of code to be executed whenever the value of a variable changes
  - Optionally can get the old value with `{oldValue}`
What is a Replace Trigger?

```javascript
var x = 10;
println("----x = {x}"');

// Defines a password variable and attaches a replace trigger to it; when the
// password changes, the trigger prints out a message reporting its new value:
var password = "foo" on replace oldValue1 {
  println("\n----ALERT! Password has changed!");
  println("----Old Value: {oldValue1}"');
  println("----New Value: {password}"');
  x++;
};
println("----x = {x}"');  // 11

// Change the value of the password variable. The trigger
// will be executed again.
password = "bar";
println("----x = {x}"');  // 12
```
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
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.
JavaFX Architecture

JavaFX Script Software

- Project Scene Graph
- Effects

Java 2D

Graphics hardware

Models a JavaFX GUI
Scene Graph: Group

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 that can be used to "chain" effects

  - `javafx.scene.effect` package API.
  - All of the core filter effect classes extend the abstract `javafx.scene.effect.Effect` base class.
Effects:
DropShadow
Example: DropShadow class

- *DropShadow* class provides 5 variables
  - 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
  }

  ...
}
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.

```xml
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
Effects:
PerspectiveTransform
PerspectiveTransform Class

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

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 – \(1\text{ms}, 1\text{s}, 1\text{m}, 1\text{h}\)
  - Eg. `var runFor = 500\text{ms}`
Two Types of Animation in JavaFX

• Transition
  > “Precanned” animation
  > Single purpose

• Animation
  > More flexible but more code
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();
  }
}
```

Click princess to do RotateTransition!
Path Transition

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();
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: Interpolator Based

```javascript
var t = Timeline {
    keyFrames : [
        KeyFrame {
            time: 0s
            values: [ tx => 0.0 ]
            action: function() { ... }
        },
        KeyFrame {
            time: 10s
            values: [
                tx => 700 tween Interpolator.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
            ]
        }
    ]
};
```

```javascript
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
    on_MouseClicked: 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,]
        }
    ]
}
```
Demo:

Building “Picture Display” Step by Step
Demo:

Building “Under the Sea” Step by Step
Media
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
JavaFX Technology
Overview