

# JavaFX™ 1.0 In Practice



Richard Bair  
Jasper Potts  
Martin Brehovsky  
Sun Microsystems



# Overall Presentation Goal

Learn what JavaFX can do for you  
(and your customers)

# Our qualifications

- Richard, Jasper, and Martin are all on the JavaFX SDK team
- Martin lead the team responsible for the JavaFX Production Suite
- Richard is one of the API leads for JavaFX SDK
- Jasper is one of the key engineers on the JavaFX SDK



JavaFX is the new client stack for graphical Java applications across all devices.

# Agenda

- Introduction to JavaFX
- JavaFX Script
- Scene Graph
- Music and Movies
- Animations and Transitions
- Tool Support
- Q & A

# Introduction to JavaFX™

- Common APIs across devices
- Scales from small devices to powerful desktops
- Brings rich media and graphical APIs to Java
- Simplifies building graphical consumer and enterprise applications

# JavaFX Script

- Expression language
- Declarative and Procedural
- Integrates with Java
- Loosely based on JavaScript

# DEMO

Hello World



# Hello World Code

```
println("Hello World")
```

# Hello World Explained

- JavaFX Script source files are called “scripts”
- Scripts which expose no public API may be “loose”
- Everything\* in JavaFX Script is an expression
  - All blocks are expressions and the last line is the result
  - The result of the println expression is null
- The JavaFX String primitive is `java.lang.String`

# Primitive Data Types

- Boolean
- Integer
- Number
- String
- Duration
- Primitives cannot be null\*

# Declaring Strings

```
var s = "Hello World";
```

```
var s2 = 'I can use single quotes too';
```

```
var i = 10;
```

```
var s3 = "The number is {i}";
```

```
var even = i mod 2 == 0;
```

```
var s4 = "The number is {if (even) then 'even' else 'odd'}";
```

```
var s5 = "The hex code for 1024 is {%x 1024}";
```

```
var s6 = "You can declare multiline strings"  
        "Just like this. Notice no '+' sign";
```

# Declaring Durations

```
var d = 23s;  
var d2 = 2.3ms;  
var d3 = 25.5 * 1.13s;  
var d4 = 5m;  
var d5 = 10h;
```

# Declaring Sequences

- JavaFX has “Sequences”, not arrays
  - A Sequence is an immutable ordered list of non-null elements
  - JavaFX supports sequence comprehensions
  - Sequences can be “sliced”
  - Sequences can optimize memory usage

# Declaring Sequences

```
// A sequence of all the positive non zero integers. This  
// actually uses very little memory since the range is  
// remembered, not the values (since sequences are  
// immutable)
```

```
var ints = [1..java.lang.Integer.MAX_INT];  
println(sizeof ints);
```

```
// creates a subsequence which contains all positive non  
// zero even integers
```

```
var even = ints[n | n mod 2 == 0];
```

```
// a simple sequence of strings
```

```
var names = ["Jasper", "Richard", "Martin"];
```

# Declaring More Sequences

// A sequence of one can omit the brackets

```
var names:String[] = "Richard";
```

// Empty sequences are not null

```
var names:String[];
```

```
println(sizeof names); // prints 0
```

// elements are accessed by index

```
var names = ["Jasper", "Richard", "Martin"];
```

```
var martin = names[2];
```

// and you can populate a sequence using a for loop

```
var hellos = for (i in [1..3]) { "Hello #{i}" }
```

# Modifying Sequences

// Inserting items into a sequence

```
var names:String[] = "Richard";  
insert "Jasper" into names;
```

// Inserting before a certain index

```
insert "Martin" before names[1];
```

// Inserting after a certain index

```
insert "Duke" after names[1];
```

// Deleting from the sequence

```
delete "Duke" from names;
```

# Operators

- $+ - / *$
- $++ --$
- $*= /= += -=$
- and or not
- $= == !=$
- mod

# Flow Control

- `if ( booleanExpression) then a else b`
- `if (booleanExpression) a else b`
- `if (booleanExpression) { a } else { b }`
- `while (booleanExpression) { ... }`
- `for (i in sequence) { ... }`
  - Can get index of item “i” by “indexof i”
- **break**
- **continue**

# Flow Control Expressions

- if ( booleanExpression) then a else b
  - An if expression returns either “a” or “b” as the result
  - If “a” or “b” is a block, then it returns the last line of the block that was selected by the if statement
- While and For are expressions which return a sequence filled with the result of the last line of the block

# Declaring Variables

- name:type style syntax
- Type inference
- var for variables, def for definitions

# Declaring Variables

// I cannot be changed. I'm inferred to be a Number.

```
def pi = 3.141592;
```

// I can be changed. And I've been explicitly declared

// as a Number

```
var radius: Number = 7;
```

// same as above except by type inference it thinks it is an

// Integer

```
var radius2 = 7;
```

# Declaring Functions

- name(args):type style syntax
- Type inference
- Must use keyword **function**
- Must use keyword **override** when overriding a function
- May be public, package, protected, or implicitly private
- May be anonymous (ie: closure)

# Declaring Functions

```
// A private function which returns the min of the two args
function min(first:Number, second:Number):Number {
    if (first <= second) then first else second
}
```

```
// a function variable and anonymous function declaration
// and function invocation
```

```
var onError:function(e:Exception):Void;
```

```
onError = function(e:Exception):Void {
    e.printStackTrace();
}
```

```
onError(e);
```

# Declaring Classes

- Classes extend classes or interfaces
- Scope modifiers: public, protected, package
  - Implicitly “script private”
- `init { }` and `postinit { }` blocks take place of Constructors
- Functions and variables declared outside a class are static
- Multiple classes may be defined in a single Script

# Declaring Classes

// This class extends a Java interface

```
public class FooModel extends TableModel { ... }
```

// This class defines two variables and a function

```
public class Location {  
    public var x:Number;  
    public var y:Number;
```

```
    public function move(newX:Number, newY:Number):Void {  
        x = newX;  
        y = newY;  
    }  
}
```

# Variable Scope Modifiers

- Variables use scope modifiers for read & write access
  - public
  - protected
  - package
  - “script private” by default
- Variables can have additional modifiers to modify write access
  - public-read
  - public-init

# Variable Scope Modifiers

```
public class ImmutableFoo {  
    // This variable can be initialized by anyone, but  
    // modified only by this script  
    public-init var name:String;  
}
```

```
public class ReadOnly extends {  
    public var x1:Number;  
    public var x2:Number;  
    // This variable can only be read by everyone, it cannot  
    // be initialized or mutated outside this script  
    public-read var distance:Number;  
}
```

# Class Initialization

- Variables are initialized from base class to child class in the order in which they are declared
- Variables are only initialized once
- `init { }` blocks are called after all variables in the class have been initialized
  - Parent class `init { }` blocks are called before child `init { }`
- `postinit { }` blocks are called after the `init { }` block
  - Parent class `postinit { }` blocks are called after child `postinit { }` blocks

# Variable Initialization Order

```
public class Parent {  
    public var name = "Richard";  
  
    init {  
        println("name is {name}");  
    }  
}  
  
public class Child extends Parent {  
    override var name = "Luke";  
}
```

# Variable Initialization Order

```
// prints out "Richard"  
var parent = Parent { };
```

```
// prints out "Luke"  
var child = Child { };
```

# Triggers

- Variables can have **on replace** triggers to execute procedural code when the bound variable is updated
- Triggers execute in an undefined order
- Variables can have multiple triggers
- Triggers are fired for **every** set, including initialization

# Triggers

```
public class Location {  
    public var x:Number on replace {  
        println("New X Value is {x}");  
    }  
  
    public var y:Number on replace {  
        println("New Y Value is {y}");  
    }  
}
```

# Bind

- **bind** is a way to tie the value of one variable to the value of an expression
- binding must be defined when the variable is initialized
- bindings are statically compiled
- bound variables cannot be set manually
- use **bind with inverse** for bidirectional binding

# Simple Binding Example

```
public class Distance extends {  
    public var x1:Number;  
    public var x2:Number;  
    // Whenever x2 or x1 changes, distance will be updated  
    // Binding can be used for invariants  
    public-read var distance:Number = bind x2 - x1;  
}
```

# Object Literals

- Concise declarative syntax for object creation
- Similar to JavaScript
- Combine with binding for maximum effect
- variable: initial-value
  - initial-value is an expression

# Object Literal

// creates a Rectangle

// x: 10 is not an assignment, it is an initialization!

```
var rect = Rectangle {  
    x: 10  
    y: 10  
    width: 100  
    height: 100  
}
```

# Nesting Object Literals

// creates a Rectangle with a Color for its fill

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

# Vars in Object Literals

// Notice that I can declare a var and use it in the literal

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

# Vars in Object Literals

// A variation that allows me to reference the color later

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

# Object Literal With If

```
// Here I'll choose the color to use based on some boolean
```

```
var highContrast = false;
```

```
var rect = Rectangle {
```

```
  x: 10
```

```
  y: 10
```

```
  width: 100
```

```
  height: 100
```

```
  fill: bind if (highContrast) then BLACK else GRAY
```

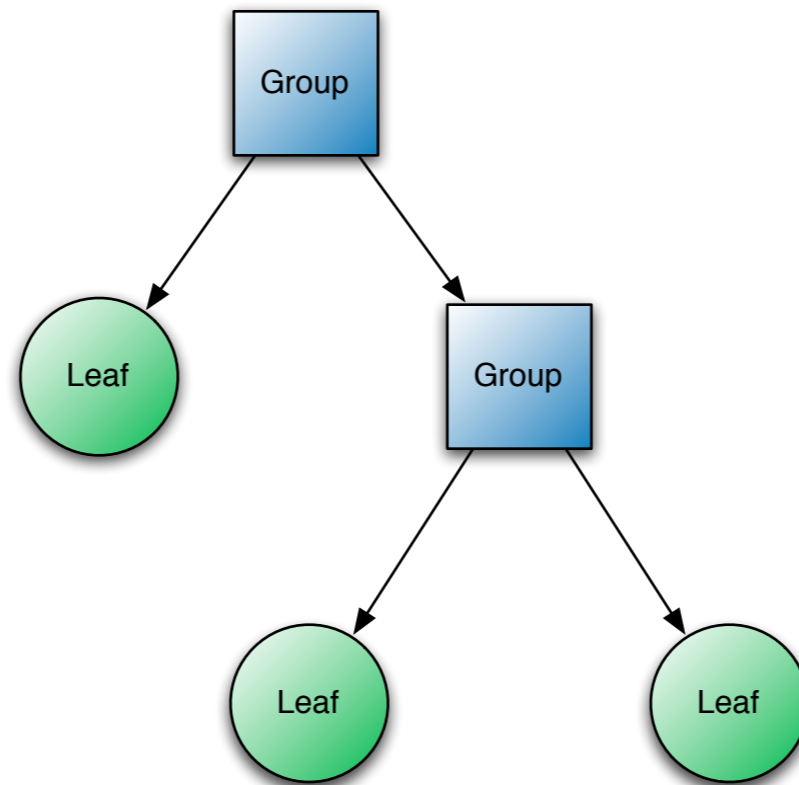
```
}
```

```
highContrast = true;
```

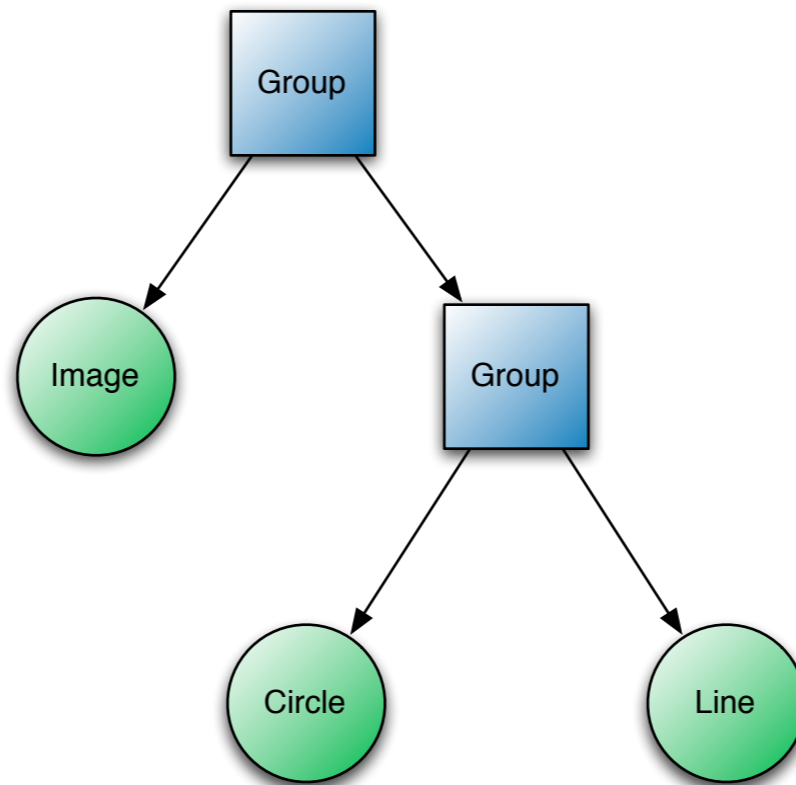
# Scene Graph

- Describes the graphics and controls in a scene
- Each node in the graph has a single parent
- Special “group” nodes have zero or more children
- “leaf” nodes have no children
- Graph is set on a Scene
- Scene is set in a Stage

# Scene Graph



# Scene Graph



# Scene

- Canvas upon which the Scene Graph is displayed
- Can set multiple CSS Stylesheets
- Can set background color (or set to null)
- Can set canvas width / height

# DEMO

Create a Scene



# Creating A Scene

```
Scene {  
  width: 400  
  height: 400  
  background: Color.BLACK  
}
```

# Stage

- Top-level container for the scene
- Contains only one Scene
- Can set Stage width / height
- Potentially represented by:
  - JFrame on desktop
  - JApplet on web page
  - SVG player on mobile

# DEMO

Create a Stage



# Creating A Stage

```
Stage {  
  style: StageStyle.TRANSPARENT  
  scene: Scene {  
    width: 400  
    height: 400  
    background: null  
    content: Rectangle { width: 100 height 100 }  
  }  
}
```

# Nodes

- Node is the base class of the entire scene graph
- Every Node has bounds
- Every Node has transforms
  - translate
  - scale
  - rotate
  - affine
- Every Node has a parent

# Node Variables

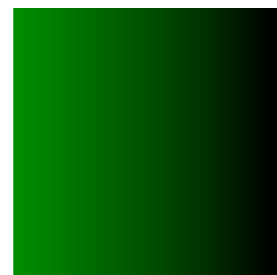
- id
- styleClass
- visible
- opacity
- focusable
- focused
- cache
- clip
- effect
- translateX, translateY
- scaleX, scaleY
- rotate
- transforms
- hover
- layoutBounds

# Node Functions

- `lookup(id)`
- `onKeyPressed`
- `onKeyReleased`
- `onKeyTyped`
- `onMouseClicked`
- `onMouseDragged`
- `onMouseEntered`
- `onMouseExited`
- `onMouseMoved`
- `onMousePressed`
- `onMouseReleased`
- `onMouseWheelMoved`

# Groups

- Have zero or more children as “content”
- Can specify a blend mode
- The children are composited within the group according to the blend mode



Top Source



Bottom Source



None



OVERLAY

# Custom Node

- Primary method of Scene Graph encapsulation
- All other nodes are not extendable
- Simply override the `create()` method

# Half-round Rectangle

// A Rectangle with a round top and square bottom

```
public class HalfRoundRectangle extends CustomNode {  
    protected override function create():Node {  
        Group {  
            content: [  
                Rectangle {  
                    width:50, height:35, arcWidth:12, arcHeight:12  
                }  
                Rectangle { y:20, width:50, height:30 }  
            ]  
        }  
    }  
}
```

# Shapes - Building Blocks

## Basic Shapes

- Arc
- Circle
- Ellipse
- Line
- Path
- Polygon
- Rectangle

## Common Variables

- stroke
- strokeWidth
- fill
- smooth

# Circle

- $x, y, \text{ radius}$
- Circles are centered about the point defined by  $(x, y)$

# Rectangle

- x, y, width, height
- Use arcWidth, arcHeight to make a rounded rect
- Rectangles are positioned with the top left corner at (x, y)
-

# Path

- Built up by various PathElements
  - LineTo
  - MoveTo
  - CurveTo
  - CubicCurveTo
  - etc

# Colors

- Colors
  - 150+ built in colors (all the HTML & CSS built in values)
  - `Color.web("#aabbcc")`
  - `Color.web("blue")`
  - `Color.rgb(128, 222, 21)`

# Linear Gradients

- startX, startY, endX, endY
  - Define the direction of the gradient
  - On the unit square
- Stops define each step in the gradient. Each stop
  - Has an offset between 0...1
  - Has a Color

# DEMO

## Create An App (Part 1)



# Images

- ImageView node shows images
- Image represents an in memory Image
- Image can load images in FG thread or BG thread
- Both ImageView and Image can scale
  - Preserve ratio
  - Fit within a specific width/height
  - When fit on Image level, keeps smaller image in memory

# DEMO

## Create an App (Part 2)



# Text

- `x, y, TextOrigin`
- By default, text positioned such that `(x, y)` is left baseline
- Fonts can be specified on `Text`
- Favor “fill” over “stroke”
- Supports multiline text
- Use alignment to align multiline text
- To center text, compute the center via layout bounds

# Text Box

- Used for text input
- Use CSS to style the TextBox
- “text” is changed to reflect the actual text in the TextBox
- “value” is changed when the text is “committed” via ENTER, TAB, etc
- “action” function is invoked when ENTER pressed
- “columns” specifies the preferred width based on the font size and number of characters to display

# Swing

- Use wrapped Swing components
  - JButton
  - JComboBox
  - etc
- Wrap any Swing component
  - `SwingComponent.wrap(someSwingComponent)`

# Effects

- Effects are placed on Nodes
- Many standard built in
  - DropShadow
  - ColorAdjust
  - GaussianBlur
  - Glow
  - Reflection
  - more...

# DEMO

## Create an App (Part 3)



# Media

- JavaFX supports both visual and audio media
- Cross platform JavaFX Media file (fxm, mp3)
- Also plays native formats (mov, wmv)
- Media class represents a media file
- MediaPlayer plays a Media file
- MediaView is the Node which displays the Media
- No built in Movie playing Control

# Summary

- JavaFX 1.0 SDK provides the APIs necessary for building creative applications
- Future updates to JavaFX will come regularly and quickly
  - 1.1 release around February 2009
  - 1.5 release around June 2009
- JavaFX 1.0 Production Suite allows you to use professional graphics in your apps

# Concluding statement

Try JavaFX Today

# Q&A



Thanks for your attention!

<http://javafx.com>