JavaFX: New Kid on RIA Block

Sang Shin
Sun Microsystems
javapassion.com
Topics

- What is and Why JavaFX?
- Key features of JavaFX
- Things you can build with JavaFX
- JavaFX script overview
- Declarative GUI building
- Scene graph
- Animation
- Media
- Deployment
- JavaFX production suite
- JavaFX Mobile
- Web services
What is & Why JavaFX?
Rich Clients Are Changing the Game!

- Clients are becoming visually rich
  - Too much engineering effort to create using traditional tools
  - Challenging the conventional notion of GUI toolkits
    - Buttons with image → Image buttons (HTML) → ???
- Clients are omnipresence
  - The concept of one software on one computer is dying...
  - Browsers are no longer the only client - They are used as delivery vehicles
- Clients are designed rather than programmed
  - Working together with graphic designers to conceptualize the interface and use cases
What does RIA mean today?

Extend RIA, across multiple screens
JavaFX Vision

JavaFX is **THE** platform for creating and delivering Rich Internet Applications (RIA) across all the screens of your life.

*JavaFX is Powered by Java*
Targeting Developer/Designer Workflow

### Creative Community
- Karl May (Mashup author/builder)
- Sean Wani (Graphic Designer)
- Saloni Sarin (Creative Director)
- Wayne Stidolph (Programmer)
- Tom Hammer (Media Editor)
- Adam Nielson (Web Master)

### Consumers
- Livleen (Student)
- Amy Lewis (Suburban wife/mom)
- Samir Arora (Business Professional)

Rich Internet Apps + content
Across desktop, mobile, TV, car
Key Features of JavaFX
Key Features

• One-stop shop RIA platform for all screens:
  > Build engaging visual experiences across desktop, browser and mobile with a unified development and deployment model.

• Broadest market reach:
  > Distribute RIAs easily across billions of devices with the power of Java.

• Designer-developer workflow:
  > Dramatically shorten your production cycle for design and development.
Key Features (Continued)

• Break free from the browser:
  > Drag-and drop a JavaFX application from the browser to deploy to the desktop.

• Built over powerful Java runtime:
  > Leverage the extreme ubiquity, power, performance and security of the Java runtime.

• Java technology compatibility:
  > Preserve your investment by enabling the use of any Java library within a JavaFX application.
Things You Can Build with JavaFX
3-D Display Shelf With the PerspectiveTransform

• The *PerspectiveTransform* built into JavaFX can be used to easily create 3-D effects
Flying Saucer

• The new out-of-process Java applet plugin in Java SE 6 update 10 enables you to make applets which can run outside of the browser.
Simple Video Player

- Incorporating video in your application is as simple as creating an instance of this component, setting a few variables and including a link to your video source.
VideoCube

- Shows how to use JavaFX technology to rotate a cube that displays video on its faces. The sample enables the user to click a cube face to play video, or drag a face to rotate the cube
Demo:

JavaFX Sample Apps
from javafx.com
JavaFX Platform
Architecture
JavaFX Platform Architecture

Applications
Content
Services

JavaFX

Application Framework
Desktop Elements
Mobile Elements
TV Elements
Common Elements
JavaFX Runtime

Java Virtual Machine

Tools
Designer Tools
Developer Tools
## JavaFX roadmap

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Product</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Product Line</td>
<td>JavaFX Desktop</td>
<td>(Winter 2008)</td>
</tr>
<tr>
<td>Mobile Product Line</td>
<td>JavaFX Mobile</td>
<td>(Spring 2009)</td>
</tr>
<tr>
<td>TV Product Line</td>
<td>JavaFX TV</td>
<td>(Summer 2009)</td>
</tr>
<tr>
<td>Other Platforms</td>
<td>With Partner platforms/OSs</td>
<td></td>
</tr>
</tbody>
</table>
JavaFX Script
Overview
What is JavaFX Script?

- JavaFX Script source files are called “scripts”
- Everything in JavaFX script is an expression
  > All blocks are expressions
  > The last line is the result
Features of JavaFX Script

• Declarative, statically-typed scripting language
• Facilitates rapid GUI development
• 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
• Cool, interesting language features for building RIA apps
  > Object literal, Sequence, Data binding, Animation, Media
Class Definition

- Address class definition: The Address class declares street, city, state, and zip instance variables all of type String

```swift
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.

```
Address {
    street: "1 Main Street"; // separated by semi colons
    city: "Santa Clara";
    state: "CA";
    zip: "95050";
}
```
Declaring Object Literals

• When declaring an object literal, the instance variables may be separated by commas or whitespace, as well as the semi-colon

• The following declaration is also correct:

```javascript
Address {
    street: "1 Main Street" // separated by whitespace
    city: "Santa Clara"     // separated by whitespace
}
```

```javascript
Address {
    street: "200 Pine Street", // separated by comma
    city: "San Francisco",     // separated by comma
}
```
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";
    }
}
```
What is a Sequence?

- In addition to the five basic data types, the JavaFX Script programming language also provides data structures known as sequences.
- 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
  > `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
Specifying Sequence's Type Explicitly

- You can also explicitly specify a sequence's type by modifying its variable declaration to include the name of the type followed by "[]":

  ```
  var weekDays: String[] = ["Mon","Tue","Wed","Thu","Fri"];
  ```

- This tells the compiler that the `weekDays` variable will hold a sequence of Strings (as opposed to a single String).
Sequences with Shorthand Notation

• There is also a shorthand notation that makes it easier to create sequences that form an arithmetic series.

• To create a sequence consisting of the numbers 1 through 100, use the following:

> var nums = [1..100];
Creating Sequences with Predicate

- You can use a boolean expression, or a **predicate**, to declare a new sequence that is a subset of an existing sequence. For example, consider the following:

  var nums = [1,2,3,4,5];

- To create a second sequence (based on items found in this first sequence) but containing only numbers greater than 2, use the following code:

  ```javascript
  var numsGreaterThanTwo = nums[n | n > 2];
  // Select all items from the num sequence where the value of an item is greater than 2 and assign those items to a new sequence called numsGreaterThanTwo.
  ```
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
```
Expression in JavaFX Script

- Expressions are pieces of code that evaluate to a result value, and that can be combined to produce "bigger" expressions.

- The JavaFX Script programming language is an expression language, which means that everything, including loops, conditionals, and even blocks, are expressions.

- In some cases (such as while expressions) the expressions have Void type, which means they don't return a result value.
Types of Expression

• Block expression
• if expression
• Range expression
• for expression
• while expression
• break and continue expression
• throw, try, catch, and finally expression
Block Expression

• A block expression consists of a list of declarations or expressions surrounded by curly braces and separated by semicolons.

• The value of a block expression is the value of the last expression.

  > If the block expression contains no expressions, the block expression has Void type.
  > Note that var and def are expressions.
Example: Block Expression

```javascript
var nums = [5, 7, 3, 9];
var total = {
    var sum = 0;
    for (a in nums) { sum += a; }
    sum;
}
println("Total is \{total\}."); // Total is 24.
```
Using Declarative Syntax (for Creating GUI)
Example of JavaFX Application

```java
import javafx.scene.paint.Color;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

Stage {
    title: "My circle"
    width: 100
    height: 100
    scene: Scene {
        content: [
            Circle {
                centerX: 50,
                centerY: 50
                radius: 40
                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

http://www.javapassion.com/handsonlabs/javafx_lang/#Exercise_1
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,
http://www.javapassion.com/handsonlabs/javafx_guibasics/index.html#7.1

DropShadow with Binding,

EffectsPlayground
http://javafx.com/samples/EffectsPlayground/index.html
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);
        },
    ]
}
JavaFX
GUI Basics II:
Interaction,
Transformation,
Binding,
Drag and drop,
Swing components
Topics

• Interaction
• Transformation
• Binding
• Drag and drop
• Swing components
Interactions
Handling Events

• All nodes have either mouse or keyboard events
  > Override the appropriate method
• Mouse events – onMouseXXXX()
  > XXXX = Entered, Exited, Pressed, Dragged, Moved, Clicked, Released, WheelMoved
• Keyboard events – onKeyboardXXXX()
  > XXXX = Pressed, Released, Typed
• Indicate interactivity by changing cursor
  > Set the cursor attribute
Example: Handling Events

- Mouse events change the color of an rectangle

```javascript
var rectangle:Rectangle = Rectangle {
    x: 20, y: 10
    width: 150, height: 70
    arcWidth: 50, arcHeight: 50
    fill: Color.LIGHTBLUE
    stroke: Color.ROYALBLUE
    strokeWidth: 3
    onMouseEntered: function( e: MouseEvent ):Void {
        rectangle.fill = Color.WHITESMOKE;
    }
    onMouseExited: function( e: MouseEvent ):Void {
        rectangle.fill = Color.LIGHTBLUE;
    }
}
```
Demo: Simple Sketch

gui2_interaction_Sketch_basic_3steps
http://www.javapassion.com/handsonlabs/javafx_guibasics2/#1.3
Transformation
Transformation class

- Provides functions to perform
  - Rotating
  - Scaling
  - Shearing
  - Translation
Rotate Transformation

- Rotates coordinates around an anchor point

```java
Stage {
    title: "Welcome to JavaFX!"
    scene: Scene {
        width: 200
        height: 200
        content: [
            Rectangle {
                transforms: Rotate {
                    angle: bind angle
                    pivotX: 10
                    pivotY: 10
                }
            }
            ...
        ]
    }
}
```
Scale Transformation

- Scales coordinates by the specified factors

```java
Stage {
    title: "Welcome to JavaFX!"
    scene: Scene {
        fill: Color.LIGHTBLUE
        content: [
            Group {
                translateX: 55
                translateY: 10
                content: [
                    Circle {
                        ...
                    },
                    Text {
                        content: "Duke"
                    },
                    ImageView {
                        ...
                    }
                ]
                transforms: bind Transform.scale(scale, scale)
            }
        ]
    } // Group
} // content
```
Shear Transformation

• Shears coordinates by the specified multipliers

Stage {
  title: "Transformation - Shear"
  scene: Scene {
    content: [
      Rectangle {
        transforms: Shear {
          x: bind shearX
          y: bind shearY
        }
        x: 40
        y: 10
        width: 100
        height: 50
        fill: Color.GREEN
        onMouseEntered: function( e: MouseEvent ):Void {
          shearX = -0.8;
        }
        onMouseExited: function( e: MouseEvent ):Void {
          shearX = -0.35;
        }
      }
    ]
  }
}
Translation Transformation

- Translates coordinates by the specified factors

```javascript
Stage {
    title: "Transformation - Translate"
    width: 100  height: 100
    scene: Scene {
        content: [
            Rectangle {
                transforms: Translate {
                    x: bind translateX
                    y: bind translateY
                }
                x: 10
                y: 10
                width: 20
                height: 20
                fill: Color.GREEN
                onMouseEntered: function( e: MouseEvent ):Void {
                    translateX = 20;
                    translateY = 30
                }
                onMouseExited: function( e: MouseEvent ):Void {
                    translateX = 0.0;
                    translateY = 0.0;
                }
            }
        ]
    }
}
```
Demo: Transformation

www.javapassion.com/handsonlabs/javafx_guibasics2/#Exercise_2
Bindings with GUI Objects
Binding with GUI objects

- The power of binding really shows when it is used with GUI objects
  > GUI objects can automatically change their shape, characteristics, and behavior
Binding with GUI objects

Stage {
    title: "DropShadow effect to a rounded Rectangle"
    resizable: false
    scene: my_scene = Scene {
        width: 300 height: 180
        fill: Color.WHITE
        content: [
            Text {
                font: Font { size: 16 }
                // Compute the center x and y coordinates of the text
                x: bind (my_scene.width - wording.length()*number_of_pixels_per_character)/2
                y: bind my_rectangle.y/2

                // Use different wording and color
                content: bind wording
                fill: bind color
            }
        ]
    }
    my_rectangle = Rectangle {
        effect: DropShadow {
            radius: bind radius
        }
        x: 50
        y: 70
        width: bind my_scene.width - 100
        height: bind my_scene.height - 90
        arcWidth: 40
        arcHeight: 40
        fill: Color.RED
        onMouseEntered: function( e: MouseEvent ):Void {
            radius = 20.0;
            wording = "Move mouse outside of the rectangle!";
            color = Color.RED;
        }
        onMouseExited: function( e: MouseEvent ):Void {
            radius = 1.0;
            wording = "Move mouse over the rectangle!";
            color = Color.GREEN;
        }
    }
}
Demo: Transformation

gui2_binding_DropShadow_Mouse
www.javapassion.com/handsonlabs/javafx_guibasics2/#Exercise_3
Drag and Drop
Mouse Pointer Locations

- The mouse (pointer's) location is available relative to several coordinate systems:
  - x,y - relative to the origin of the MouseEvent's node,
  - sceneX,sceneY - mouse location relative to the origin of the Scene that contains the node,
  - screenX,screenY - relative to origin of the screen that contains the mouse pointer,
  - dragX, dragY - if the MouseEvent is part of a press-drag-release gesture, the relative to the location of the press event, otherwise 0.
Drag and Drop (using dragX, dragY)

- Drag an object around the screen

```javascript
var sx:Number = 0; var ex:Number = 0;
var sy:Number = 0; var ey:Number = 0;
var rectangle:Rectangle = Rectangle {
  x: bind ex y: bind ey
  width: 150 height: 70 arcWidth: 50, arcHeight: 50
  fill: Color.LIGHTBLUE
  stroke: Color.ROYALBLUE strokeWidth: 3
  cursor: Cursor.HAND
  onMousePressed: function( e: MouseEvent ):Void {
    sx = e.dragX + ex;
    sy = e.dragY + ey;
  }
  onMouseDragged: function( e: MouseEvent ):Void {
    ex = e.dragX + sx;
    ey = e.dragY + sy;
  }
}
```
Drag and Drop (using dragX, dragY)

- Mouse pointer location information

```plaintext
****onMousePressed:  e.sceneX = 4.0, e.dragX = 0.0, sx = 0.0, ex = 0.0
-----onMouseDragged: e.sceneX = 5.0, e.dragX = 1.0, sx = 0.0,, ex = 1.0
-----onMouseDragged: e.sceneX = 6.0, e.dragX = 2.0, sx = 0.0,, ex = 2.0
-----onMouseDragged: e.sceneX = 7.0, e.dragX = 3.0, sx = 0.0,, ex = 3.0
-----onMouseDragged: e.sceneX = 8.0, e.dragX = 4.0, sx = 0.0,, ex = 4.0
****onMousePressed:  e.sceneX = 8.0, e.dragX = 0.0, sx = 4.0, ex = 4.0
-----onMouseDragged: e.sceneX = 8.0, e.dragX = 0.0, sx = 4.0,, ex = 4.0
-----onMouseDragged: e.sceneX = 8.0, e.dragX = 1.0, sx = 4.0,, ex = 5.0
-----onMouseDragged: e.sceneX = 10.0, e.dragX = 2.0, sx = 4.0,, ex = 6.0
****onMousePressed:  e.sceneX = 80.0, e.dragX = 0.0, sx = 6.0, ex = 6.0
-----onMouseDragged: e.sceneX = 81.0, e.dragX = 1.0, sx = 6.0,, ex = 7.0
-----onMouseDragged: e.sceneX = 82.0, e.dragX = 2.0, sx = 6.0,, ex = 8.0
-----onMouseDragged: e.sceneX = 83.0, e.dragX = 3.0, sx = 6.0,, ex = 9.0
-----onMouseDragged: e.sceneX = 84.0, e.dragX = 4.0, sx = 6.0,, ex = 10.0
-----onMouseDragged: e.sceneX = 85.0, e.dragX = 5.0, sx = 6.0,, ex = 11.0
```
Demo: Drag and Drop

www.javapassion.com/handsonlabs/javafx_guibasics2/index.html#Exercise_4
Swing Components
Swing Components

- SwingButton
- SwingCheckBox
- SwingComboBox
- SwingLabel
- SwingList
- SwingRadioButton
- SwingScrollPane
- SwingSlider
- SwingTextField
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
  > \textit{RotateTransition} – rotation
  > \textit{FadeTransition} – opacity
  > \textit{TranslateTransition} – move a node along a straight line
  > \textit{PathTransition} – move an object along a defined path
  > \textit{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

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

http://www.javapassion.com/handsonlabs/javafx_animation/#Exercise_1
http://www.javapassion.com/handsonlabs/javafx_animation/#Exercise_2
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
    radius = 300
}

How the value changes over time
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,]
        }
    ]
}
```
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

http://www.javapassion.com/handsonlabs/javafx_animation/index.html#4.1
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
FX Media API Overview

- Simple model-view-controller design
- All classes in `javafx.scene.media` package.
- `MediaView` takes all cool features of SceneGraph node, such as effects, transforming, clip, opacity, etc...
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
Media Format Support

• Platform specific media types are supported by native platform
  > DirectShow on Windows (wmv, wav, avi, asf, etc...)
  > Quicktime on Mac (mov, mp4, etc)
• FXM is the cross platform format that can be played across desktop computers and mobile phones.
• FXM is an FLV subset which currently consists of
  > VP6 Video Encoding
  > MP3 Audio Encoding
• On2 FLX tool available for transcoding.
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

http://javafx.com/samples/SimpleVideoPlayer/index.html
Deployment
Deployment Options

• JavaFX 1.0 applications can be deployed using the two standard Java deployment technologies
  > Java Plugin: A tool used for deploying Java applets that run inside a web browser
  > Java Web Start: A tool used for deploying stand-alone Java applications on the desktop, using JNLP (Java Network Launching Protocol).

• Or using mobile emulation
  > JavaFX 1.0 Mobile Emulator Beta Release: A tool provided with the JavaFX 1.0 SDK, which displays your applications as they would look on a typical mobile device.
New Plugin Architecture

• Live outside the browser
  > Cannot crash the browser
• Applets can run in separate JVM
  > Choose the version they want
  > Configurable per applet basis
  > Live beyond the browser
• Unify Applet and JavaWeb Start model
• Works on FF3 and IE7 only
Execution Models

- Standard, Web Start, Applet, Mobile emulator
Demo:

Draggable Applet
JavaFX Production Suite
What is JavaFX Production Suite?

- Is a set of tools that enables designers and developers to work independently on graphics and application logic to develop expressive content for rich Internet applications (RIAs) and mobile devices.
  - Designers create the graphics and convert them from Adobe® Photoshop or Adobe illustrator to JavaFX format, or they can convert SVG graphics to JavaFX format.
  - Application developers can manipulate the graphic objects as independent objects and apply filters, effects, and animation.
- A single graphics file can provide all of the artwork for the application.
What Tools Does JavaFX Production Suite Contain?

- JavaFX 1.1 Plugin for Adobe® Illustrator
- JavaFX 1.1 Plugin for Adobe Photoshop
- JavaFX 1.1 Media Factory
  > JavaFX Graphics Viewer: Displays any JavaFX-format graphic.
  > SVG Converter: Converts files from SVG format to JavaFX format
Why Separation of Roles between Designers and Developers

- Combining graphics directly with code is great for small projects and encapsulated components.
- However - for large applications (complex graphics, many screens, bigger teams) separation of graphics and code is required.
- Designers and developers need to work on graphics and code as separate parts in independent iterations.
- The tooling needs to assure a seamless integration of the both parts at any time.
Roles when developing rich applications

• Both designer and developer have the same goal – develop the cool looking application
  > But use different tools and approach/view of the problem
  > Both fill in different pieces
  > Iteration is frequent
  > We need to make sure the integration of the different pieces is as easy as possible

• Designer (Interaction Designer)
  > Responsible for creating the complete UI experience
  > Delivers graphics, media, ...
  > Makes sure the application is usable

• Developer
  > Focuses on delivering business logic
  > Makes sure the application does what it needs to
Typical Rich Application Development Context

Developer

IDE

create

Any development environment that is able to produce JavaFX script code, for example NetBeans or IntelliJ IDEA

Designer

Graphics tool

create

Either vector or bitmap oriented authoring tool, for example Adobe Illustrator, Photoshop, etc

Application

BusinessLogic uses Graphics

<<source files>>

<<component>>

<<artifact>>
Typical Rich Application Development Workflow

1. Content authoring
2. Content and business logic iterations
3. Final application
4. Deployment

- Graphic prototype
- Business logic development

Flow Chart:
- Content authoring to Content and business logic iterations
- Content and business logic iterations to Final application
- Final application to Deployment
- Deployment to Graphic prototype
- Graphic prototype to Business logic development
- Business logic development to Content authoring
Typical Rich Application Development Workflow

1. Designers and developers meet to **draw a graphic prototype and agree on the graphic objects and their names**. This step is very important, because it ensures that the designer can drop the graphic into the application when it is ready without causing rework.

2. The designers create the graphic content using their design tools, while the developers create the business logic of the application, using the rough sketch as a placeholder and the graphic object names for their object IDs.

3. The content design and business logic go through as many iterations as necessary.

4. When both the content and business logic of the JavaFX application is final, it is deployed.
Anatomy of a JavaFX Content File (FXZ)

- FXZ is a compressed file using zip compression and file format. It always contains an FXD file, which is a text description of the graphic. In addition, it might contain embedded assets, such as image files or True Type fonts. The archive can be easily compressed and uncompressed using a zip utility.

- You can view the image contained in the JavaFX Content File (FXZ) in either the JavaFX Graphics Viewer (one of the Production Suite tools) or the NetBeans IDE.
Anatomy of a JavaFX Content File (FXD)

• Graphic objects from the original graphic (layer items and attributes) are described in the JavaFX Data file (with an FXD extension)

• FXD is a textual format using the same object literal syntax as JavaFX Script

• Easy to edit in visual tools – a round trip is possible

• Allows storing metadata
FXD Explained

Developer

IDE

JavaFX plugin

Any development environment that is able to produce JavaFX script code, for example NetBeans or IntelliJ IDEA.

create

Designer

Graphics tool

FXD plugin

Either vector or bitmap oriented authoring tool, for example Adobe Illustrator, Photoshop, etc.

create

Application

JavaFX Script

FXD

Key nodes

UI Stubs

The UI stubs connect together the key nodes in the graphic content with application business logic.

uses

references

The nodes in FXD tree with unique IDs to be referenced from the business logic.
UI Stub Files for Developers

- Developers can use the NetBeans IDE to generate a file that extends the UiStub class and declares the variables for all objects in the JavaFX Data File (FXD) and assigns the ID values to them.
FXD Example

Group {
  id: "face"

  content: [
    Circle { id:"background" centerX:40 centerY:40 radius:39 fill:Color.YELLOW stroke:Color.BLACK strokeWidth:3.0},
    Circle { centerX:25 centerY:30 radius:5 fill: Color.BLACK},
    Circle { centerX:55 centerY:30 radius:5 fill: Color.BLACK},
    Line{ startX:32 startY:23 endX:16 endY:15 stroke:Color.BLACK strokeWidth:4.0},
    Line{ startX:45 startY:23 endX:61 endY:15 stroke:Color.BLACK strokeWidth:4.0},
    QuadCurve { id: "mouth" stroke:Color.BLACK strokeWidth:3.0 fill: Color.TRANSPARENT
      startX:20 startY:60 endX:60 endY:60 controlX:40 controlY:80
    }
  ]
}
Demo: Designer/Developer Workflow

http://www.javapassion.com/handsonlabs/javafx_productionsuite/#Exercise_1
JavaFX Mobile
Mobile vs. Desktop

- Common profile
- Different implementation
- Platform specific APIs

> Performance
> Capabilities
JavaFX Mobile Application Architecture

JavaFX Application

JavaFX Application Runtime

JavaFX Language Runtime

CLDC + MIDP + MSA
JavaFX Mobile Application - Packaging

JavaFX code

JavaFX compiler

Java code

Java compiler

Preverification

JAR (CLDC ready code)

Static code analysis performance and size optimizations

Optionally bundle JavaFX Platform Runtime
JavaFX Mobile Architecture

JavaFX Application Runtime

<table>
<thead>
<tr>
<th>Scenegraph</th>
<th>Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX backend</td>
<td>Media backend</td>
</tr>
<tr>
<td>JSR 226 (SVG)</td>
<td>JSR 135 (MMAPI)</td>
</tr>
<tr>
<td>JSR 184 (Mobile3D)</td>
<td>JSR 234 (AMMS)</td>
</tr>
<tr>
<td>JSR 239 (OpenGL ES)</td>
<td>Native VP6 codec</td>
</tr>
</tbody>
</table>

CLDC + MIDP
JavaFX API vs. Mobile JSRs

• Java APIs are available from JavaFX
• Common APIs for common functionality
• Platform specific APIs for specific functionality
• Mobile specific APIs
  > SMS send and receive
  > Embedded camera access
  > Persistent data storage
  > Contact list access
Deployment Model

• OTA (Over-The-Air)
  > Possible to prototype on in-the-market phones
  > Lower performance
  > Bigger storage footprint (bigger downloads)
  > Demonstration purposes only

• Embedded
  > Better platform integration possibilities
  > Ability to leverage all target device capabilities (i.e. hardware graphics acceleration)
  > Better performance
  > Smaller storage footprint
  > Better developer experience
Demo:
JavaFX Mobile

http://javafx.com/samples/SimpleVideoPlayerMobile/index.html
JavaFX Technology
Overview