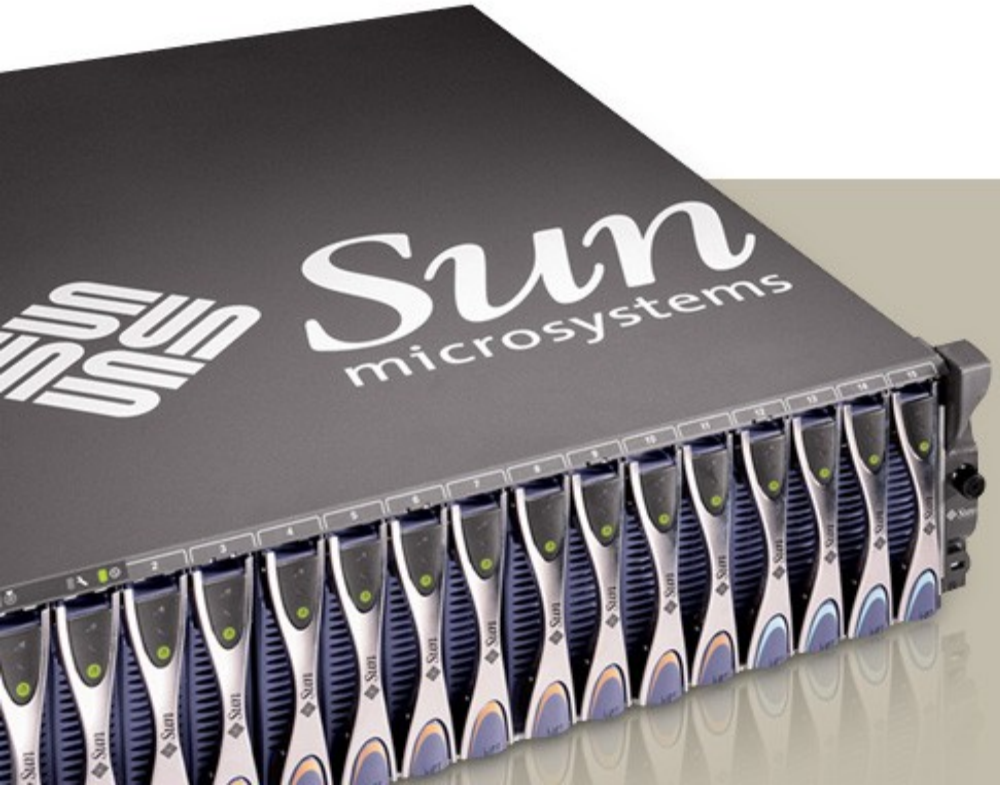
JavaFX - Rich GUIs Made Easy

SUN TECH DAYS 2008–2009
A Worldwide Developer Conference

Angela Caicedo
Technology Evangelist

Overview of the JavaFX SDK

Overview



Rich Clients Are Changing the Game

- Client are becoming visually rich
  - > Too much engineering effort to create these using traditional tools
  - > Challenging the conventional notion of GUI toolkits
- Clients are omnipresence
  - > The concept of one software on one computer is dying...
  - > Browsers are no longer the only client
- Clients are designed rather than programmed
- Working together with graphic designers to conceptualize the interface and use cases

JavaFX Vision*

JavaFX is THE platform for creating and delivering Rich Internet Applications across all the screens of your life

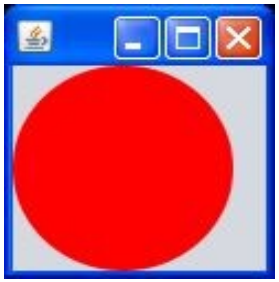
* the marketing...

JavaFX Script Programming Language

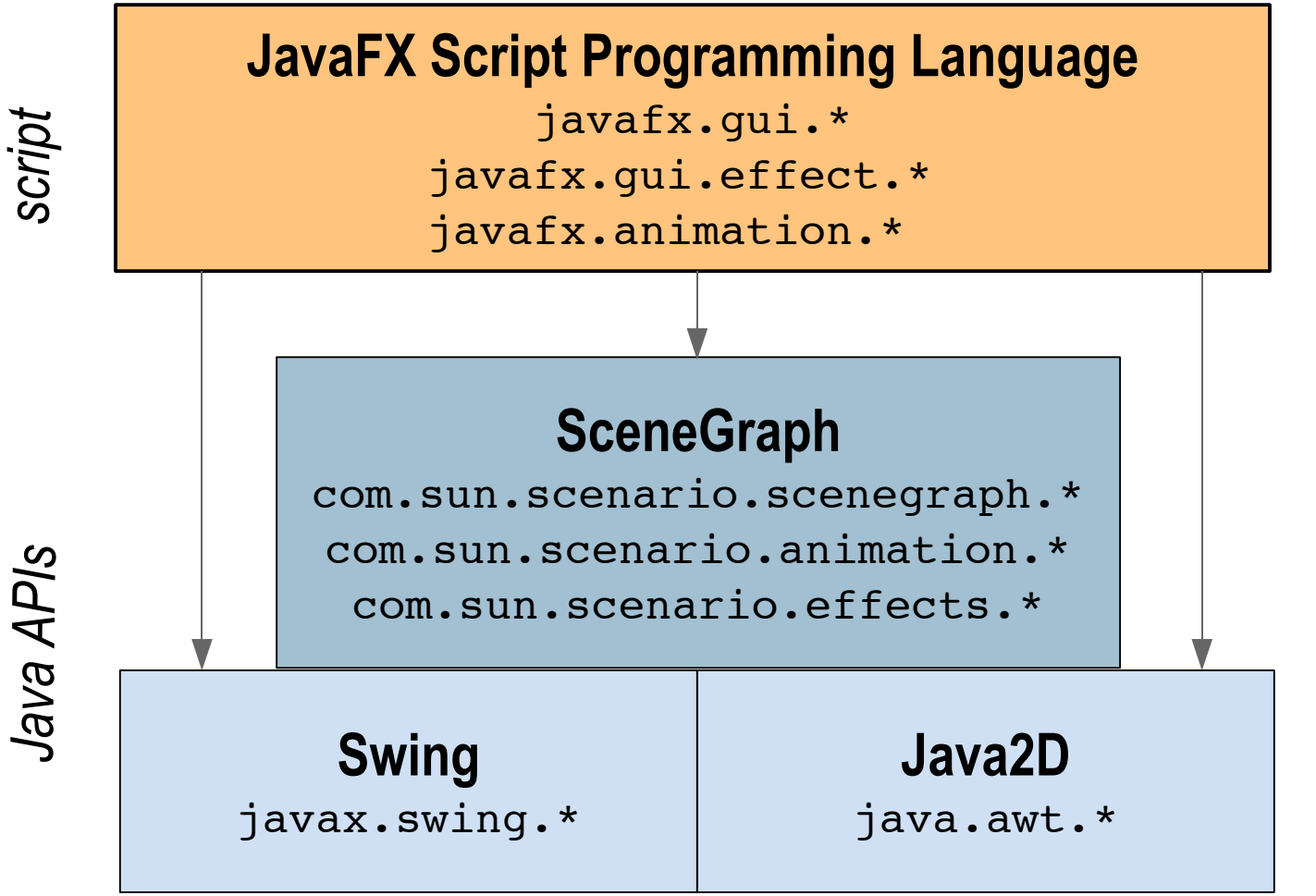
- Declarative, statically-typed scripting language
- Facilitates rapid GUI development
- Many cool, interesting language features
- Runs on Virtual Machine for the Java™ platform
- Deployment options same as Java programs
- Fully utilizes Java class libraries behind the scenes
- For content designers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

Example of JavaFX Application

```
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;


Stage {
    scene: {
        content: [
            Circle {
                centerX: 50
                centerY: 50
                radius: 50
                fill: Color.RED
            }
        ]
    }
}
```



JavaFX Technology Stack



Note: JavaFX Script programs can call any Java APIs

(Some) Language Features

Basic JavaFXScript Class

Syntax is Java-like with shades of JavaScript

```
class HelloWorldNode extends CustomNode {
    public var text:String;
    public override function create(): Node {
        return Text {
            x: 10, y: 50
            font: Font {
                size: 50
            }
            content: bind text
        }
    };
}
```

Declarations

- Variables `var fred:Number;`
- Constants `def PI:Number = 22 / 7;`
- Access modifiers
  - Default access – script only, no modifier specified
  - **`package`** – same package
  - **`protected`** – same package or subclases
  - **`public`** – any class, any script, any package
  - **`public-read`** – publicly readable, but writeable only from within the current script.
    - **`package public-read or protected public-read`**
  - **`public-init`** – publicly initialized. Write access as public-read

Example – Declarations

```
// file tutorial/one.fx
package tutorial;

public-read var x = 1;
```

```
// file tutorial/two.fx
package tutorial;

println(one.x);
```

Output:

1

```
// file tutorial/one.fx
package tutorial;

public-read var x = 1;
```

```
// file tutorial/two.fx
package tutorial;

println(one.x);
one.x = 2;
```

Compile time error

Example – Declarations

```
// Inside file tutorial/one.fx
package tutorial;
public class one {
     public-init var message;
}
```

```
// Inside file two.fx
import tutorial.one;
var o = one {
     message: "Initialized from different package!"
}

println(o.message);
```

Output:

```
Initialized from different package
```

Example – Declarations

```
// Inside file tutorial/one.fx
package tutorial;
public class one {
     public-init var message;
}
```

```
// Inside file two.fx
import tutorial.one;
var o = one {
     message: "Initialized from different package!"
}
println(o.message);
o.message = "Changing the message...";
                                        // WON'T COMPILE
println(o.message);
```

Basic Data Types

- If not specified => compiler infer the correct type
- Garden variety type
  - **`String`**
  - **`Number/Integer`** – byte, short, int, long, BigInteger
  - **`Boolean`**
  - **`Void`**
- Durations – `1ms`, `2s`, `4m`, `8h`
- Sequences – more later
- Functions

```
var doExit = function():Void {
   System.exit(0);
};
```

Sequences

```
var time:Duration[] = [60ms, 60s, 60m, 60h];
var days = [1..31];
```

- Insert, delete, membership and reverse

```
insert 5s into time;
insert "Thu" before weekDays[2];
delete 31 from days;
var revDays = reverse days;
if (!(31 in days) or (30 in days)) “February”
```

- Slice via range and predicate

```
var oddDays = days[n | (n % 2) == 1];
var days=["Mon","Tue","Wed","Thu","Fri","Sat","Sun"];
var weekdays = days[0..<5];
var weekend = days[5..];
var days2 = days[0..<];
```

Data Binding

- Associates the value of a target variable with the value of a bound expression
  - Changes to the bound expression will cause the value to be reapplied to the field

```
var r = 10;
var area = bind r * r * Math.PI;
r = 5;
area == 78.5714 //true
```

Bound Functions

- We need => Reevaluate the function if internal values changes

```
var scale = 1;
function makePoint(xt:Number, yt:Number): Point {
    return Point {
        x: xt * scale, y: yt * scale
    };
}
var x = 3;
var y = 3;
var myPoint = bind makePoint(x, y);
x = 5;
println(myPoint.x) //The value is 5
scale = 3;
println(myPoint.x) //The value is 5
```

Example – Bounded Function

- Changes to the internal values of a function will cause the entire function to be reevaluated
  - Used in conjunction with **`bind`**

```
var scale = 1;
bound function makePoint(xt:Number, yt:Number): Point {
    return Point {
        x: xt * scale, y: yt * scale
    };
}

var x = 3;
var y = 3;
var myPoint = bind makePoint(x, y);
x = 5;
FX.println(myPoint.x) //The value is 5
scale = 3;
FX.println(myPoint.x) //The value is 15
```

Binding with Sequences

- Use bind with for expressions

```
var seq1 = [1..3];
def seq2 = bind for (item in seq1) item*2;
printSeqs();
insert 4 into seq1;
printSeqs();

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

Output

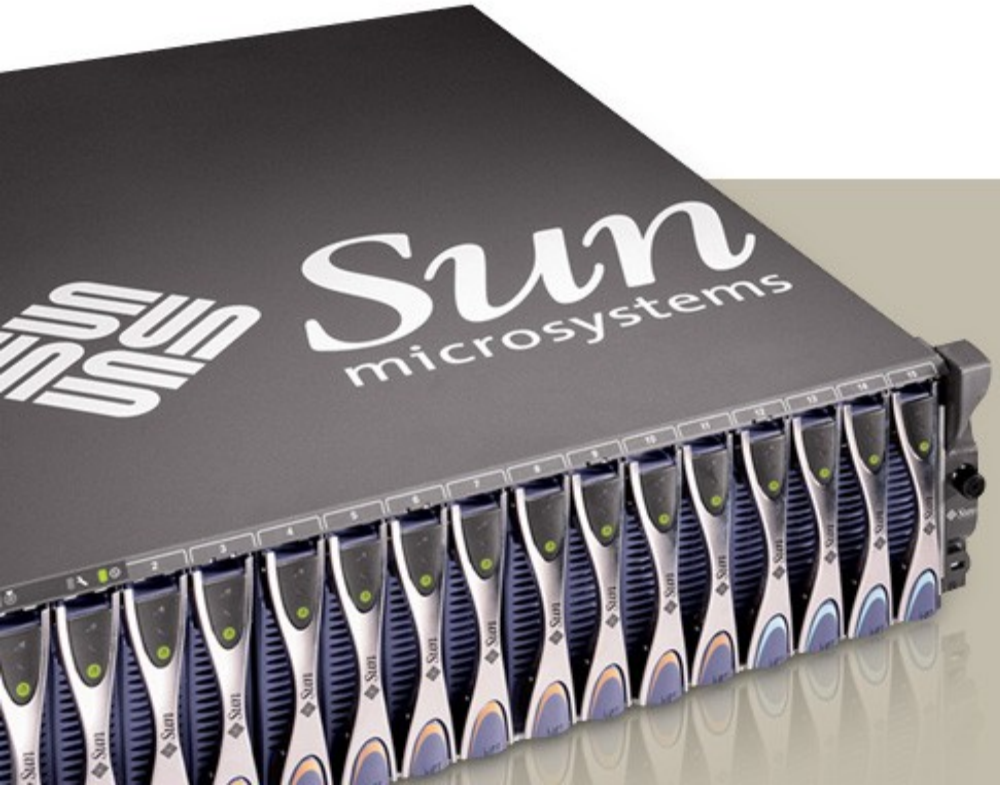
```
First Seq:
1
2
3
Second Seq:
2
4
6
First Seq:
1
2
3
4
Second Seq:
2
4
6
8
```

Triggers

- Associate a block of code to a variable
- When the value of the variable changes, the code is executed
- Similar to **`PropertyChangeListener`**

```
//oldValue is optional
var text on replace oldValue {
   println(“Old value = '{oldValue}'”);
   println(“New value = '{text}'”);
}
text = “Hello”
Old value = ''
New value = 'Hello'
```

Graphical Objects*



*The fun stuff

Base Graphical Objects

- Graphical objects
  - Text, geometric shapes, text, Swing components
- Some common attributes in nodes
  - Transformation – translate, shear, rotate, scale
  - Clip – displaying only part of the node based on a geometric shape
  - Effect – type of effect, eg. blurring, shadowing, to apply
  - Events – mouse, keyboard
  - Opacity – setting the translucency
  - List is not exhaustive

Text

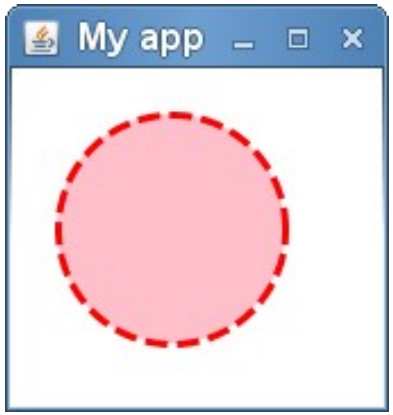
- Defines a node for displaying text

```
Text {
    effect: DropShadow {
        offsetX: -10
        offsetY: -10
    }
    font: Font {
        name: “DirtyBakersDozen”
        size: 50
    }
    fill: Color.ROYALBLUE
    stroke: Color.BLUE, strokeWidth: 3
    x: 15, y: 80
    content: "Hello World"
}
```

Geometric Shapes

- Arc, ellipse, line, polygon, circle, rectangle
- Very similar to text

```
Circle {
      centerX: 70, centerY: 70
      radius: 50
      fill: Color.PINK
      stroke: Color.RED
      strokeWidth: 3
      strokeDashArray: [ 7 ]
      strokeDashOffset: 2
}
```

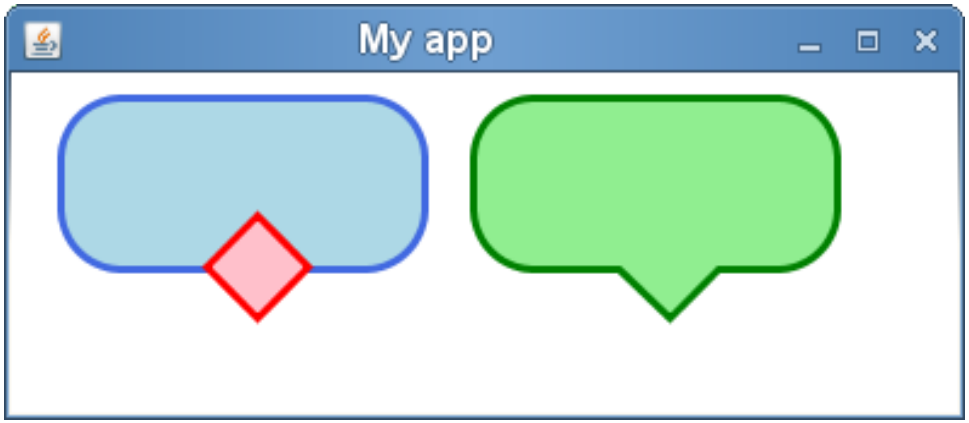


Custom Shapes

- Two ways of defining custom shapes
  - Combining existing shapes
  - Drawing a totally new shape
- Combine existing shape with `ShapeIntersect` or `ShapeSubtract`
  - Either add or subtract from shape
- Draw new shapes with `Path` and path elements
  - Path elements include `MoveTo`, `ArcTo`, `HLine`, `VLine`, `QuadCurve`, etc.
- Can be manipulated like a regular geometric shape

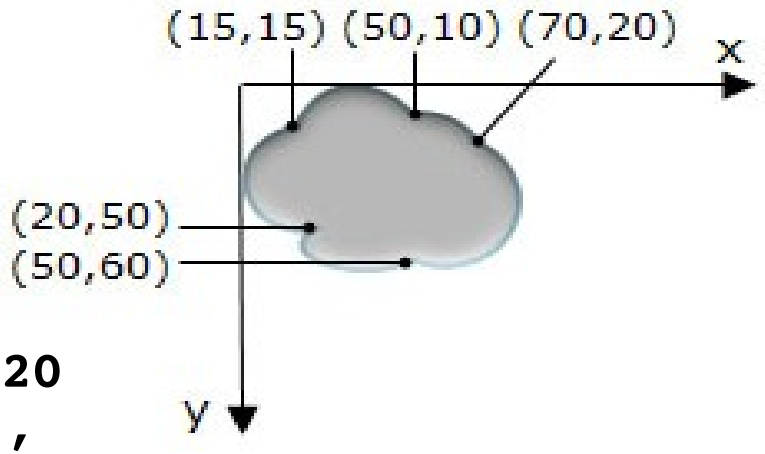
Example – ShapeIntersect

```
var rectangle = Rectangle {
    x:10 y:20 width:140 height:70
    fill:Color.LIGHTBLUE stroke:Color.BLUE
    arcHeight:20 arcWidth:20 strokeWidth:3}

var diamond = Polygon {
    points:[90,90, 110,70, 130,90, 110,110 ]
    fill:Color.LIGHTPINK  stroke:Color.RED
    strokeWidth: 3}

var newShape = ShapeIntersect {
    translateX:170
    fill: Color.LIGHTGREEN
    stroke: Color.GREEN
    strokeWidth: 3
    //Union of the 2 shapes
    a: [rectangle diamond ]
}
```



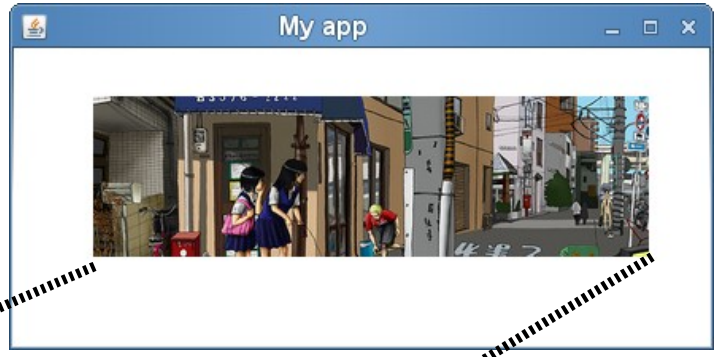
Example – Path

```
Path {
    fill: Color.LIGHTGRAY
    stroke: Color.GRAY
    strokeWidth: 3
    elements: [
      MoveTo { x: 15 y: 15 },
      ArcTo { x: 50 y: 10 radiusX: 20
       radiusY: 20 sweepFlag: true},
      ArcTo { x: 70 y: 20 radiusX: 20
       radiusY: 20 sweepFlag: true},
      ArcTo { x: 50 y: 60 radiusX: 20
       radiusY: 20 sweepFlag: true},
      ArcTo { x: 20 y: 50 radiusX: 10
       radiusY: 5 sweepFlag: false},
      ArcTo { x: 15 y: 15 radiusX: 10
       radiusY: 10 sweepFlag: true},
     ]
    effect: Lighting{light: DistantLight{azimuth: 90}}
}
```



Images

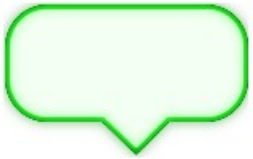
```
ImageView = ImageView {
    clip: Rectangle {
        y: 30 x: 50
        width: 350 height: 100
    }
    image: Image { url: "file:///..."}
}
```



Some Effects Supported In JavaFX

```
effect: SepiaTone { level: 0.5 }
```

```
effect: Glow { level: 0.7 }
```



Original image

```
effect: GaussianBlur {
    input: SepiaTone {
        level: 0.5 }
    radius: 10.0
}
```

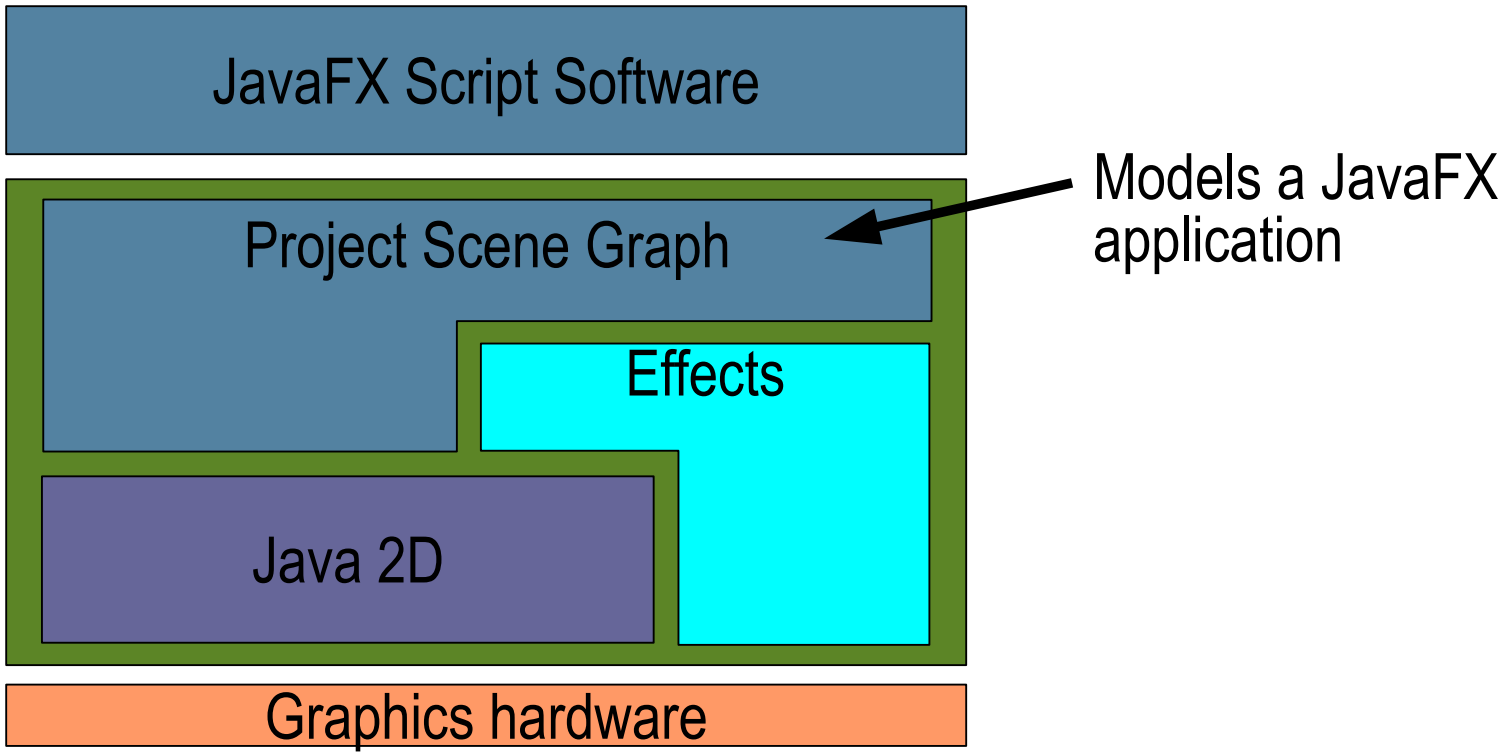
```
effect: Reflection {
    fraction: 0.7 }
```

Lighting Effect

```
effect: Lighting{
    surfaceScale: 7
    light: DistantLight{
        azimuth: 90
        elevation: 30
    }
}
```

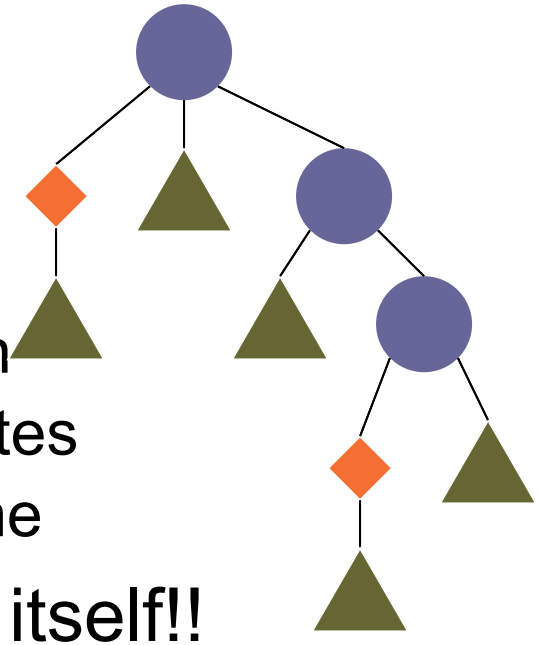
```
effect: Lighting{
    surfaceScale: 7
    light: SpotLight {
        x: 0 y :0 z: 50
        pointsAtX: 10
        pointsAtY: 10
        pointsAtZ: 0
        color: Color.YELLOW
    }
}
```

JavaFX Architecture Again



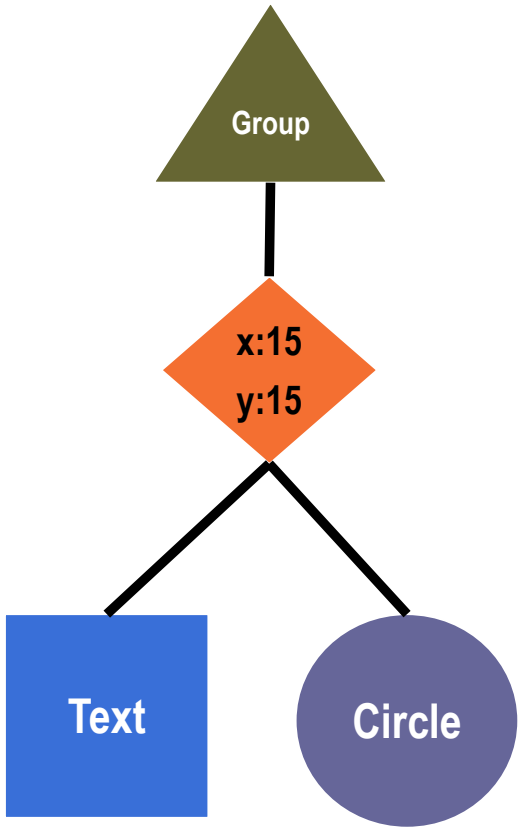
What Are Scene Graph?

- A hierarchical representation of graphical objects
  - Tree like structure
  - Basis of JavaFX graphics engine
- Scene graph elements
  - Nodes – images, text, Swing widgets
  - State – visibility, opacity, transformation
  - Events – mouse, keyboard, node updates
  - Animation – varying properties over time
- A scene graph knows how to render itself!!
  - JavaFX scene graph engine is available at http://scenegraph.dev.java.net
  - Usable from Java

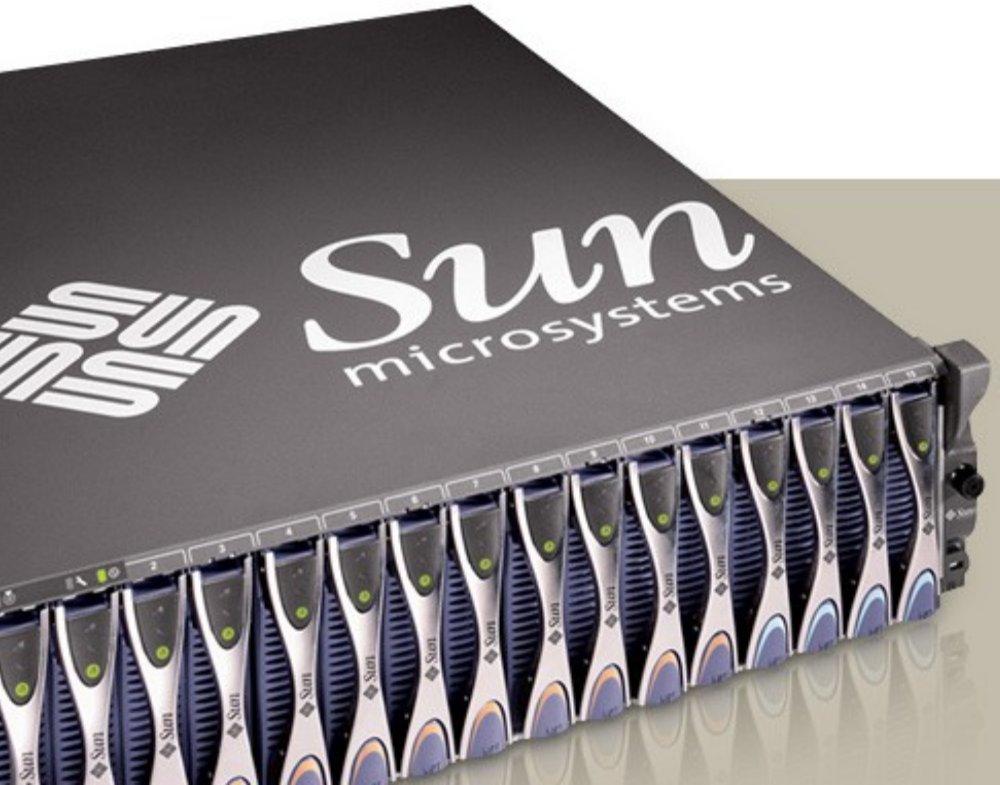


Group – Node Container

```
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
        }
    ]
}
```



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

Changing the color of the rectangle

```
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;
    }
}
```

Example – Handling Events

Dragging an object around the screen

```
var sx:Number = 0; var dx:Number = 0;
var sy:Number = 0; var dy:Number = 0;

var rectangle:Rectangle = Rectangle {
    x: bind dx y: bind dy
    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 - dx;
        sy = e.dragY - dy;
    }
    onMouseDragged: function( e: MouseEvent ):Void {
        dx = e.dragX - sx;
        dy = e.dragY - sy;
    }
}
```

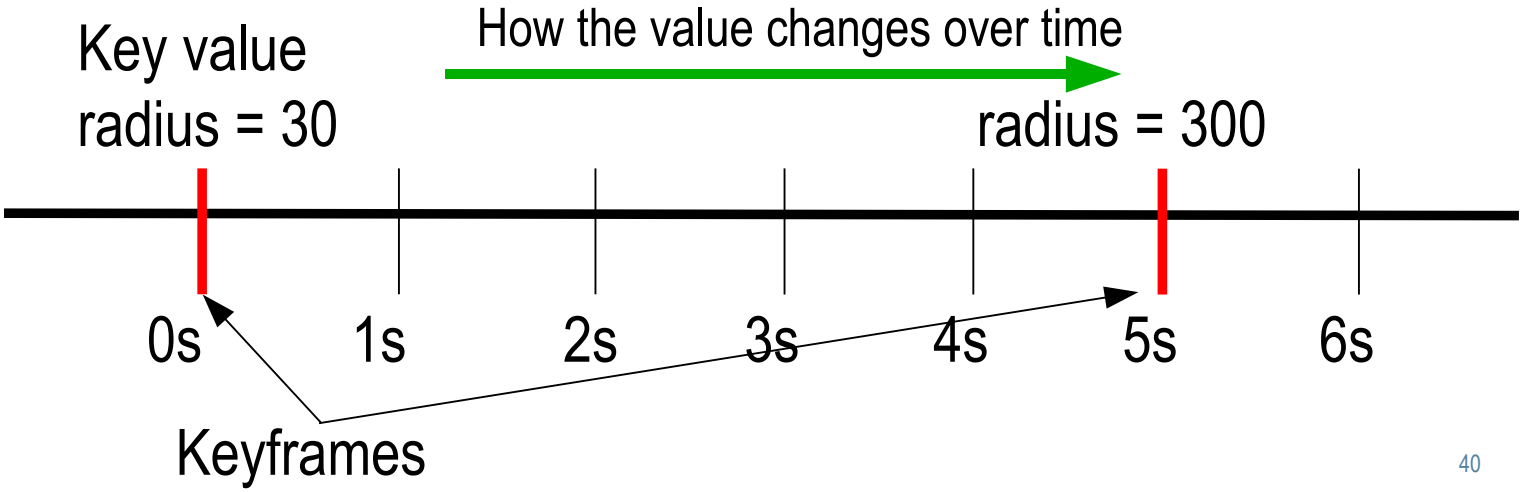
Animation

- Define several `KeyFrames` at different time intervals
  - Includes state of `KeyValues`
  - How these values are interpolated
- Assign these `KeyFrames` to `Timeline`
- Playback control

```
timeline.playFromStart()
timeline.rate = -1 //Play in reverse,
  normal speed
```

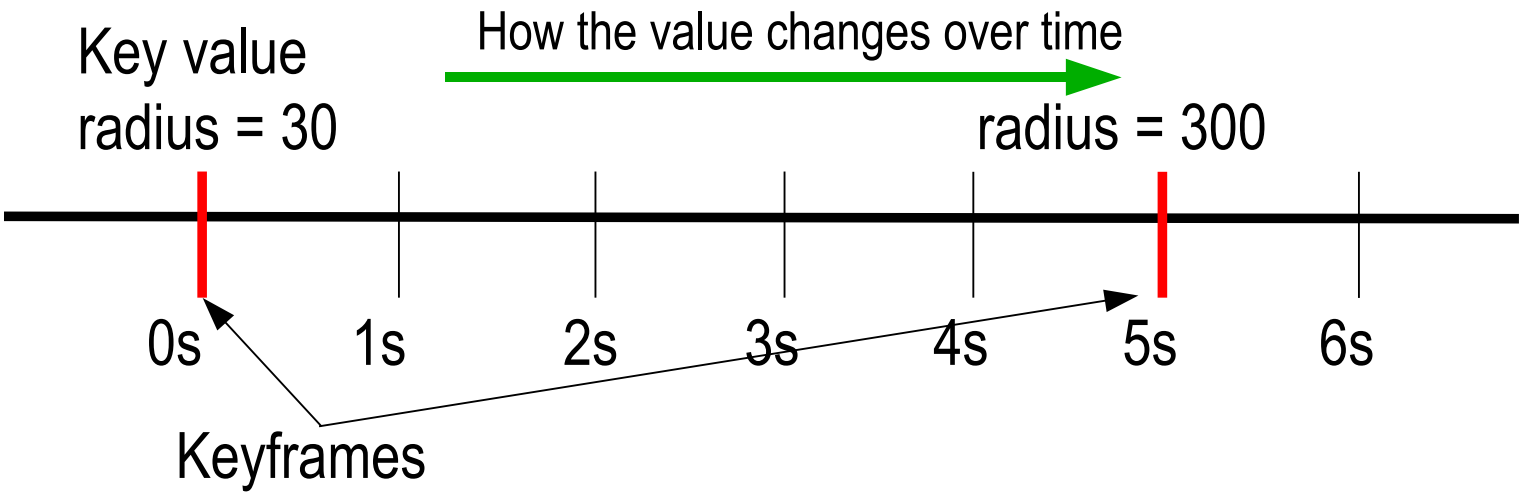
Example – Defining Key Frames

```
Timeline {
    keyFrames: [
        KeyFrame {
            time: 0s
            values: [ radius => 30 ]
          }
        KeyFrame {
            time: 5s
            values: [
                radius => 300 tween Interpolator.LINEAR
            ]
        }
    ]
}
```

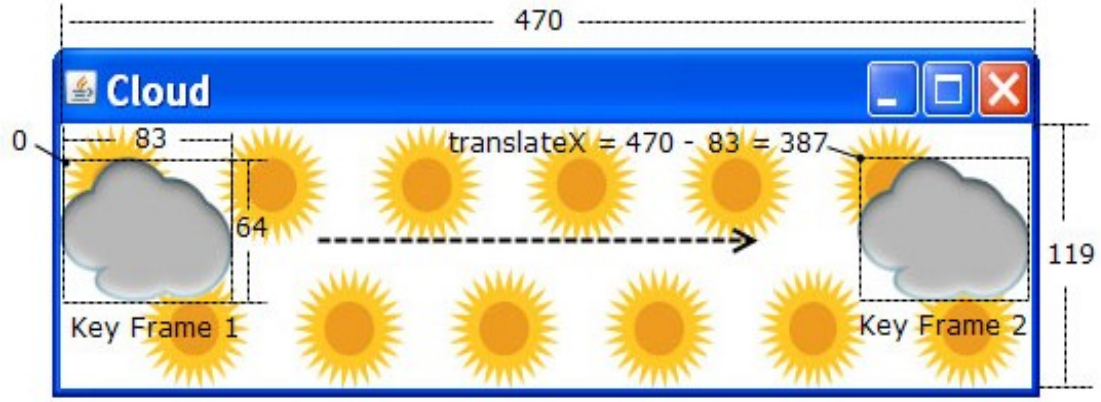


Example – Shorthand Notation

```
Timeline {
    keyFrames: [
        at(0s) {
            radius => 30
        }
        at(5s) {
            radius => 300 tween Interpolator.LINEAR
        }
    ]
}
```



Example – Animating an Object




```
var x: Number;

Timeline {
  keyFrames: [
          at (0s) {x => 0.0},
          at (7s) {x => 387.0 tween Interpolator.LINEAR}
  ]
}.play();


Path{
  ...
  translateX: bind x
  ...}
```

Data Format Parser

- Includes a 'pull' parser that supports JSON and XML
- To use in 'event' mode
  - Extends **`PullParser`**
  - Specify the format and input stream
  - Handle input as the parser returns tokens
- Can be use in 'linear' mode as well
  - Direct the parser with **`forward()`** and **`seek()`**

Example – PullParser in 'Event' Mode

```
public class TwitterParser extends PullParser {
    //Trigger when there is a token to be read
    override var event on replace {
        if (event != null) {
            if (event.type == PullParser.START_VALUE)
              System.out.println("start: name = {event.name}")
            else if (event.type == PullParser.END_VALUE)
              System.out.println("\tend: name = {event.name}")
        }
    }
}

var p = TwitterParser {
    documentType: PullParser.JSON
    input: someInputStream
}
p.parse();
```

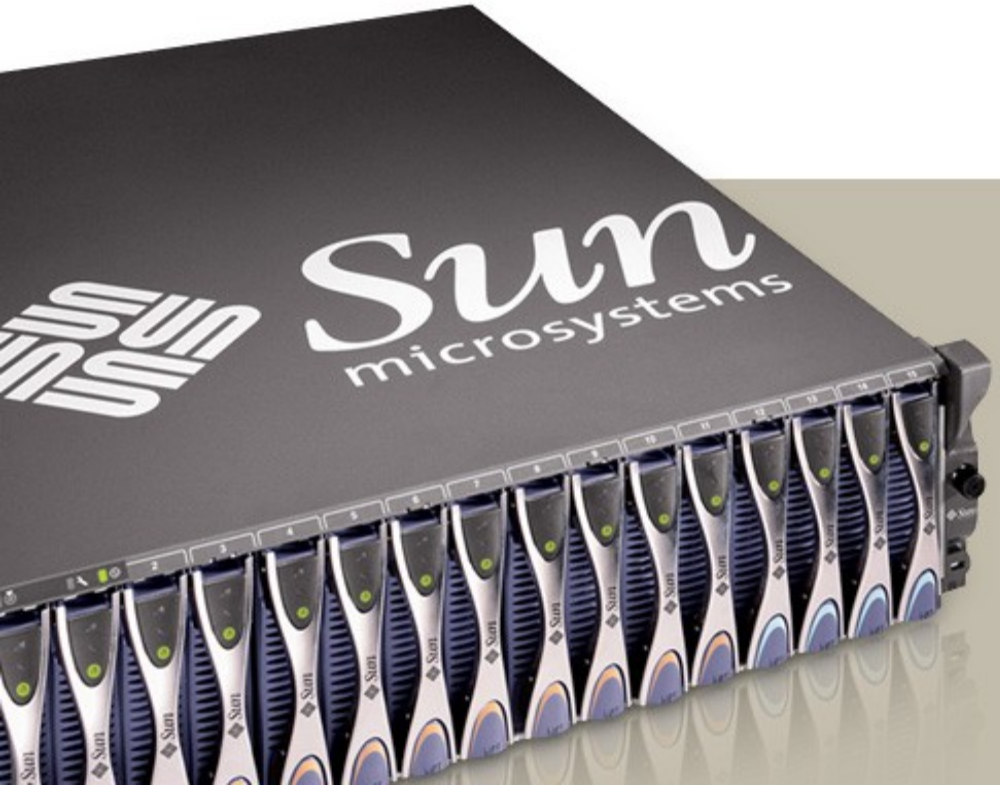
Accessing REST Resources

- Includes an asynchronous HTTP request class
- Need to specify the location and the HTTP method
- Provides lifecycle events
  - **`started`**, **`connecting`**, **`writing`**, **`reading`**, **`done`**
- Invoke `enqueue()` to start request

Example – Making HTTP Request

```
public class GetTwitter extends HttpRequest {
    override var input on replace {
        if (input != null) {
            try {
                def p = TwitterParser {
                    documentType: PullParser.JSON
                        input: input
                }
                p.parse();
            } finally {
                input.close();
            }
        }
    }
}
                GetTwitter {
                    location:
                "http://twitter.com/statuses/public_timeline.json"
                     method: HttpRequest.GET
                }.enqueue();
```

JavaFX Application



Top Level Container

- Provides an execution agnostic top level container
  - Eg. for Java – `main()`, `Applet`, `JFrame`, etc
- `Stage` is the top level container
  - Regular, Applet, JavaWeb Start, mobile
  - Defines the characteristics like title, size, location, handling exit, etc
  - Provides hooks for handling different types of deployment – eg. Applet
- `Stage` contains `Scene`
  - The 'panel' for displaying the content

Example of a Stage

```
Stage {
    title : "My app"
    extension: [
        AppletStageExtension { //Valid for applet only
            onDragStart: function(e: MouseEvent): Void {
                circle.fill = Color.LIGHTGREEN;
            }
            onDragFinished: function(e: MouseEvent): Void {
                circle.fill = Color.PINK;
            }
        }
    ]
    scene: Scene {
        width: 500 height: 300
        content: [ circle ]
    }
}
```

What Are Executed?

- `Stage` or Stage variables
  - `Stage` object
  - `Stage` variables – cannot create Stage objects in file with class definition
- `run()` function
  - Will be executed as thread

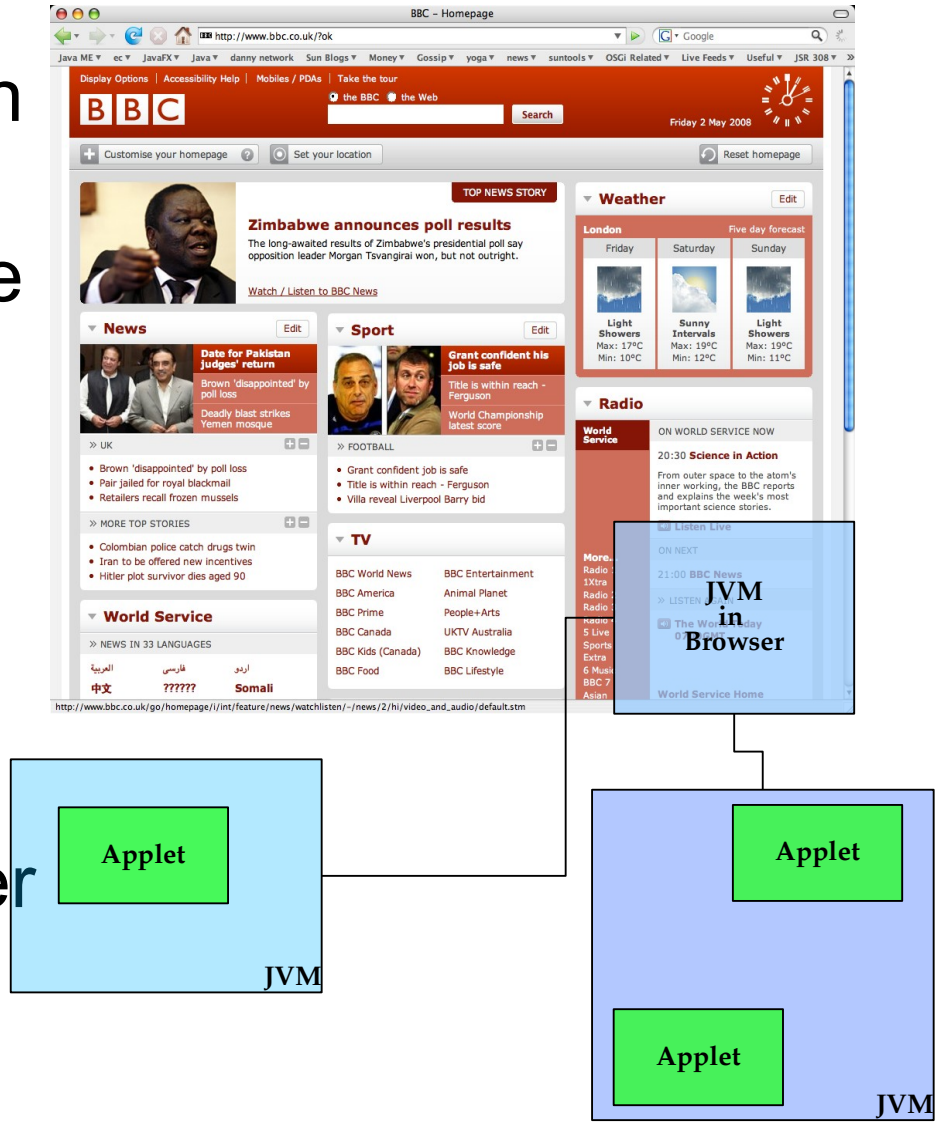
```
Main.fx
Scene {
    ...
}
function run(): Void {
    ...
}
```

Displayed

Executed as a thread

New Plugin Architecture

- Java plugin redesigned in JDK6u10
- Plugin lives outside of the browser
- Applets can be run in separate VM
  - > Can specify version
  - > Configurable per applet
  - > Live beyond the browser
- Will not crash the browser
  - > Requires FF3 or IE7



Deploying JavaFX Application As Applet

- Compile with `javafxc`
- Package with `javafxpackager`
  - Defaults to DESKTOP profile – regular, html, jnlp
- One click with NetBeans 6.5

```
<script src="http://.../javaws/dtfx.js"></script>
<script>
    javafx({
        archive: "http:.../sea.jar",
        draggable: true,
        width: 600,
        height: 300,
        code: "sea.Main",
        name: "sea"
    });
</script>
```


Sun
microsystems
THANK YOU
Angela Caicedo
Technology Evangelist
angela.caicedo@sun.com
SUN TECH DAYS 2008–2009
A Worldwide Developer Conference